VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro dynamickou tvorbu formulářů založený na ASP.NET MVC3

System for Dynamic Forms Building Based on ASP.NET MVC3

2013 Bc. Petr Šindel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Šindel**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Systém pro dynamickou tvorbu formulářů založený na ASP.NET MVC 3
System for Dynamic Forms Building Based on ASP.NET MVC 3

Zásady pro vypracování:

Systém, který bude vytvořen v rámci této práce, bude vycházet ze zkušeností se systémem pro tvorbu dynamických formulářů v Google Docs. Na základě těchto zkušeností implementuje student vlastní aplikaci v ASP.NET MVC 3. Tato aplikace bude sloužit pro snadnou správu registrací na akce nebo vytváření a vyhodnocování dotazníků. Výsledná aplikace by měl být finalizován jako krabicové řešení, s přesným konfiguračním postupem pro nasazení na serveru.

Student se v rámci práce seznámí i s implementací verze aplikace pro ASP.NET a jedním z cílů této práce je adekvátně vyhodnotit klady a zápory jednotlivých technologií a popsat možné přístupy k převodu z ASP.NET aplikace na ASP.NET MVC aplikaci.

Jednotlivé cíle diplomové práce jsou:
1. Analýza a návrh systému dynamických formulářů pro organizaci akcí.
2. Prostudování a popis technologie ASP.NET MVC 3.
3. Popis zobrazovacího enginu „Razor".
4. Srovnání technologií ASP.NET, ASP.NET MVC 2 a ASP.NET MVC 3.
5. Implementace systému v ASP.NET MVC 3.
6. Srovnání vývoje v ASP.NET a ASP.NET MVC 3 s uvedením problémů na které diplomant narazí při implementaci. V tomto bodě se předpokládá spolupráce dalším diplomantem, který se soustředí na vývoj tohoto systému v ASP.NET.
7. Implementace přihlašovacího modulu do vytvořené aplikace pomocí Windows Live, portálu Netstudent.cz a LDAP na VŠB. V případě problému vše b práci popsat a navrhnout případné řešení.
8. Popis systému a jeho nasazení.
9. Testování zátěže systému na přístup většího počtu uživatelů.
10. Zhodnocení testování a případná aktualizace kódu na základě zjištěných problémů.

Seznam doporučené odborné literatury:

ASP.NET MVC 3: http://www.asp.net/mvc/mvc3
ASP.NET Razor Tutorial: http://www.w3schools.com/razor/

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013

.....

Chtěl bych poděkovat vedoucímu mé diplomové práce Ing. Janu Martinoviči Ph.D. za odborné vedení a rady v průběhu práce.

Abstrakt

Cílem této diplomové práce je provedení analýzy a implementace systému pro tvorbu dynamických formulářů za využití technologie ASP.NET MVC3. Tento systém má umožnit jednoduchou a přehlednou správu formulářů (akcí a dotazníků), které mohou jednotliví uživatelé vytvářet a následně je zpřístupnit ostatním uživatelům k přihlášení.

Práce stručně shrnuje historii technologie MVC, kdy jsou uvedeny různé frameworky využívající tuto technologii. Hlavně však popisuje základní principy a funkce ASP.NET MVC, ve kterém je implementován samotný systém. Zvlášť jsou rozebrány a popsány různé možnosti uživatelské autorizace, které jsou využity v systému. Ta je realizována za využití integrovaného autorizačního systému, tak i pomocí VŠB LDAP, Facebooku a Windows Live. V poslední části je provedeno otestování aplikace při přístupu většího počtu uživatelů, kdy je provedena také analýza získaných údajů.

Klíčová slova: Dynamické formuláře, ASP.NET MVC3, Model-View-Controller, MVC, Razor, OAuth, Facebook, Windows Live

Abstract

The aim of this thesis is the analysis and implementation of a system for creating dynamic forms using ASP.NET MVC3. This system should allow simple and transparent administration forms (actions and questionnaires) that individual users can create and then make them available to other users to log on.

This thesis briefly summarizes the history of MVC technology and different frameworks using this technology. Mainly describes the basic principles and functions of ASP.NET MVC, in which is system implemented. Separately are analyzed and describe different types of user authentication that are used in the system. It is implemented using an integrated authorization system, VSB LDAP, Facebook and Windows Live. The last part is focused for testing with large number of users and analysis of obtained data.

Keywords: Dynamic forms, ASP.NET MVC3, Model-View-Controller, MVC, Razor, OAuth, Facebook, Windows Live

Seznam použitých zkratek a symbolů

.NET	–	.NET Framework
API	–	Application Programming Interface(rozhraní pro programování aplikací)
ASP.NET	–	Arcitecture ASP.NET(architektura ASP.NET)
ASP.NET MVC	–	Arcitecture ASP.NET MVC(architektura ASP.NET MVC)
C#	–	C# language(jazyk C#)
CGI	–	Common gateway interface(rozhraní CGI)
HTML	–	HyperText Transver Protocol(značkovací jazyk pro hypertext)
IDE	–	Integrated development enviroment(integrované vývojové prostředí)
IIS	–	Internet information services(internetová informační služba)
J2EE	–	Java 2 enterprise edition
JS	–	JavaScript
MVC	–	Model view controller
LDAP	–	Lightweight directory access protocol(protokol LDAP)
RAD	–	Rapid application development(rychlý vývoj aplikací)
REST	–	Representational state transfer
SDK	–	Software development kit(sada SDK)
SQL	–	Structured Query Language(strukturovaný dotazovací jazyk)
SSL	–	Secure sockets layer(vrstva bezpečných socketů)
SSO	–	Single sign-on(jednotné přihlašování)
TDD	–	Test-driven development(testy řízený vývoj)
UI	–	User interface(uživatelské rozhraní)
URL	–	Uniform resource locator(adresa url)
VB	–	Visual basic
XML	–	Extensible markup language(jazyk XML)
XMPP	–	Extensible Messaging and Presence Protocol(rozšiřitelný protokol pro posílání zpráv a zjištění stavu)

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Rozesílání elektronických pozvánek na akce se stalo během posledních let velice oblíbenou záležitostí. Největší zásluhu na tom mělo masové rozšíření sociálních sítí, zejména Facebooku. Uživatelé mohou velice jednoduše vytvořit akci a odeslat pozvánku uživatelům, kteří se na akci mohou přihlásit.

Vytváření těchto akcí však obsahuje celou řadu problémů a nedostatků. Mezi největší patří pokročilá správa uživatelů a otázek, na které by měl uživatel odpovědět. Například na Facebooku je možno v akci vytvořit otázky, ale pokud se přihlásí více uživatelů, dojde k posunutí dané otázky níže na stránce a přihlášený uživatel si jí nemusí vůbec všimnout. Také každý přihlášený uživatel může přidávat vlastní otázky, díky čemuž se pořadateli akce ještě více komplikuje situace.

Podobný problém nastává při vyhodnocování otázek. Pořadatel akce nemá přehled o otázkách, které vytvořil on a které vytvořil někdo jiný. Pro vyhodnocení otázek musí projít celou historii a výsledky si zapsat na papír. Další nevýhodou je nemožnost určení maximálního počtu přihlášených uživatelů a také chybí možnost vytvoření a vyhodnocení dotazníků. Posledním nedostatkem je nemožnost vložení informací o vytvořené akci na externí stránky pro informování dalších lidí.

Z těchto důvodů byl nejprve proveden návrh a následně implementace systému, který by měl popsané problémy odstranit. Systém bude umožňovat snadné a jednoduché vytváření formulářů (akcí nebo dotazníků). Každý formulář bude obsahovat základní údaje a seznam otázek, na které budou uživatelé odpovídat. Při vytváření otázek bude možnost vybrání typu otázky z několika předdefinovaných typů, díky čemuž si autor bude moci vygenerovat libovolnou otázkovou strukturu. Také bude možnost definovat grafický styl pro vložení informací o akci na externí webové stránky.

Při přihlášování k formuláři si uživatelé budou moci vybrat z několika autorizačních možností podle toho, která jim bude nejvíce vyhovovat. Samotný systém bude naprogramován pomocí ASP.NET MVC, který se v současnosti stává velmi oblíbeným a používaným programovacím nástrojem.

1.1 Struktura práce

Samotná práce se skládá z několika částí. Začátek práce, v kapitole 2, je věnován analýze systému. V následující kapitole 3 si popíšeme technologii MVC, která obsahuje vysvětlení principu fungování MVC, tak i různé MVC frameworky včetně ASP.NET MVC (kapitola 3.4). Další kapitola (4) vysvětluje nový zobrazovací engine Razor a možnosti předávání dat mezi částmi aplikace, který byl uveden s ASP.NET MVC3. Poté následují kapitoly, ve kterých jsou zmíněny rozdíly mezi jednotlivými verzemi ASP.NET MVC (kapitola 5) a srovnání vývoje aplikace mezi ASP.NET WebForms a ASP.NET MVC(kapitola 6). Následně v kapitole 7 jsou rozepsány možnosti uživatelské autorizace, které obsahují popis různých autorizačních možností implementovaných v systému. Poslední kapitola 8 je věnována možnostem testování aplikace s využitím většího počtu uživatelů, včetně vyhodnocení testu provedených v implementovaném systému.

2 Analýza systému

Tématem diplomové práce bylo navrhnout a vytvořit systém, umožňující tvorbu dynamických formulářů (akcí a dotazníků) za využití technologie ASP.NET MVC3.

Systém umožní jednoduchou a přehlednou správu formulářů, ke kterým se budou přihlašovat jednotliví uživatelé. Bude možno si zobrazit seznam přihlášených uživatelů, jejich odpovědí nebo případně nějakého uživatele z akce odhlásit.

Při vytváření formuláře je možno vytvořit seznam otázek, na které může uživatel odpovědět. Systém umožní výběr z několika předdefinovaných typů, aby bylo možno vytvářet libovolnou strukturu otázek.

Uživatelé budou mít možnost využití několika způsobů přihlášení k systému, díky tomu si budou moci vybrat způsob autorizace, který je pro ně nejlepší. K dispozici bude jak integrovaný autorizační systém, tak i přihlášení pomocí VŠB LDAP, Facebooku a Windows Live.

Pro přihlášení uživatele k formuláři bude možno využít jak vygenerovaný url odkaz, tak i iframe, který bude možno vložit na vlastní stránky. Díky tomu bude možno informovat všechny návštěvníky daných webových stránek o blížící se akci. Uživateli bude také umožněno přihlášení pomocí několika autorizačních možností.

Pro využití větším počtem lidí bude systém podporovat možnost výběru z více jazykových mutací a to z češtiny a angličtiny.

2.1 Typy uživatelů

Aplikace rozlišuje 2 typy uživatelů. Prvním je běžný uživatel a druhým je administrátor.

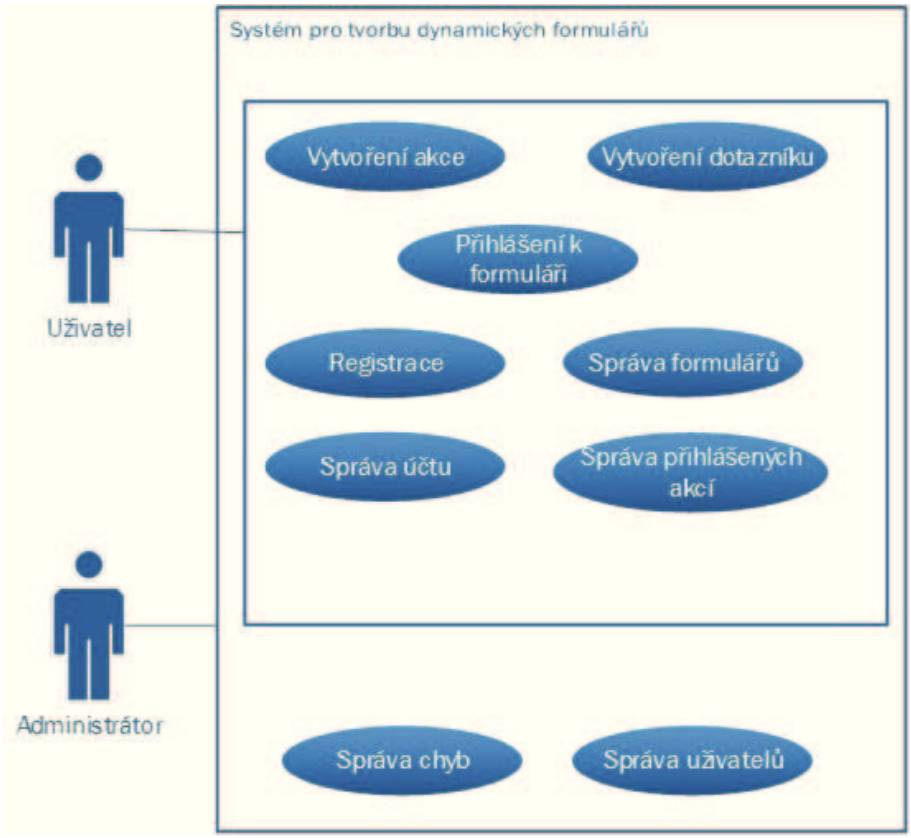
Běžný uživatel bude nejrozšířenější uživatelská role v systému. Umožňuje uživateli využívat většinu funkcí systému jako vytváření akce, přihlašování a správa vlastních akcí. Také má možnost prohlédnout si odpovědi přihlášených lidí u jednotlivých akcí. Systém také umožňuje jednoduchý export informací o akci do PDF souboru.

Administrátor má oproti uživateli možnost vidět chybová hlášení, která nastanou při chodu aplikace. Díky tomu může reagovat na vzniklé problémy a vyřešit je úpravou zdrojového kódu. Také může zvýšit práva běžného uživatele na administrátorská.

2.2 Funkce systému

Samotný systém obsahuje množství různých funkcí. Základní a nejpoužívanější funkce jsou zobrazeny na obrázku 1.

Na obrázku je uveden pouze seznam základních funkcí v jejich elementární podobě. Každá funkce obsahuje řadu kroků, které musí být provedeny pro její splnění. Administrátor má oproti běžnému uživateli přístup k dvěma dalším funkcím, a to správě chyb a uživatelů.



Obrázek 1: Funkce systému

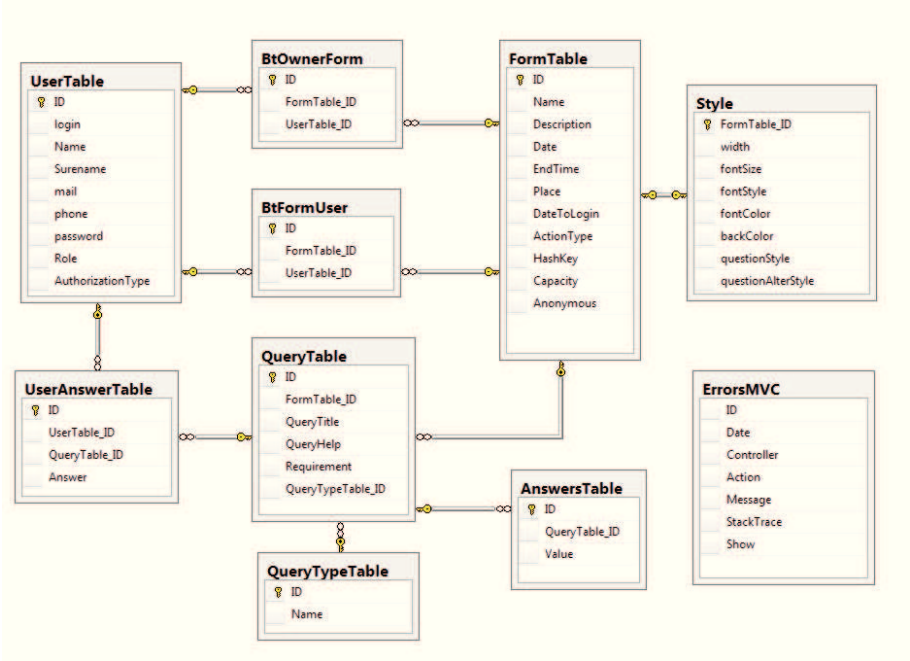
2.2.1 Konceptuální datový model

Pro ukládání dat využívá aplikace SQL databázi, jejíž struktura je ilustrována pomocí E-R diagramu na obrázku 2.

Celá SQL databáze obsahuje 10 tabulek, ze kterých jsou 2 vazební. V každé tabulce jsou uložena různá data, která jsou využívána v různých částech a funkcích aplikace. Funkce jednotlivých tabulek jsou stručně popsány v následujícím přehledu:

- **FormTable-** V této tabulce jsou uloženy všechny informace o formulářích
- **UserTable-** Tato tabulka obsahuje seznam všech zaregistrovaných uživatelů
- **QueryTable-** V této tabulce nalezneme definici jednotlivých formulářových otázek
- **AnswerTable-** Zde jsou uloženy možnosti odpovědí k dané otázce
- **UserAnswerTable-** Tato tabulka obsahuje odpovědi jednotlivých uživatelů na otázky, které jsou přiřazeny danému formuláři
- **Style-** Tato tabulka obsahuje definici stylů, jak bude vypadat přihlašovací iframe

- **ErrorsMVC-** Zde jsou uloženy chybové hlášení, které způsobily neočekáváný pád aplikace
- **QueryTypeTable-** V této tabulce jsou uloženy jednotlivé typy otázek, které je možno použít
- **BtOwnerForm-** Vazební tabulka, určující vlastníka daného formuláře
- **BtFormUser-** Vazební tabulka, obsahující informace, kteří uživatelé jsou přihlášení k formuláři



Obrázek 2: E-R diagram

2.2.2 Vytvoření formuláře

Při vytváření nového formuláře musí uživatel projít sérií kroků, které vedou k jejímu vytvoření. Nejprve je nutno zadat základní informace, jako je název, místo, datum konání a datum, do kterého se lze přihlašovat. U dotazníku je možno ještě vybrat možnost anonymního odpovídání, které nevyžaduje autorizaci uživatele.

V následujícím kroku je možno vytvořit sérii otázek, které bude muset uživatel vyplnit při přihlašování na akci. Samotný princip vytváření otázek je popsán v bodu 2.2.3

Díky výběru z několika typů otázek lze vytvořit libovolnou strukturu otázek, díky čemuž si lze formulář přizpůsobit dle aktuální potřeby. Další krok umožní uživateli upravit vzhled přihlašovacího iframu. Přihlašovací iframe umožní uživatelům vložení

informací o akci na vlastní stránky. Pomocí stylizace může upravit iframe tak, aby měl stejný vzhled jako web, na který bude vložen. Je možno definovat použitý font, šířku iframu, barvu písma i podkladu.

V posledním kroku je uživatel informován o vytvoření nové akce. Systém automaticky vygeneruje dva přihlašovací odkazy, díky kterým se může uživatel přihlásit. První odkaz využívá samotné prostředí naší aplikace. Druhý obsahuje iframe, který je možno umístit na externí stránky a z nich je možno se přihlásit na akci.

Také je vygenerován iCalendář, který je možno synchronizovat s kalendáři jako je Google, Outlook, atd.

2.2.3 Vytvoření otázek

Každý formulář (akce nebo dotazník) může obsahovat libovolný počet otázek, které lze vygenerovat z následujících předdefinovaných typů:

- Krátká textová odpověď
- Víceřádková textová odpověď
- Výběr jedné možnosti ze seznamu
- Výběr více možností ze seznamu

Při vytváření otázky je nejprve nutno zadat název otázky a pomocný text otázky a vybrat požadovaný typ. Pokud se jedná o otázku obsahující možnosti k výběru, je možno vygenerovat libovolný počet možností, ale minimálně dvě.

Následně je možno otázku vytvořit a přiřadit ji k danému formuláři.

Systém bude také podporovat možnost úpravy nebo smazání dané otázky. Úprava otázek bude umožňovat upravit informace o dané otázce. Bude možno změnit název i pomocný text a počty odpovědí. U počtu možností půjde pouze přidání dalších možností a ne jejich ubrání. Pokud budeme vyžadovat menší počet možností, tak bude třeba danou otázku smazat a vytvořit novou.

Při mazání otázky dojde nejprve k označení dané otázky ke smazání (barevnému označení). Po odeslání formuláře dojde nejprve ke kontrole, zda-li na danou otázku neodpověděl nějaký uživatel. Pokud ano, dojde nejprve ke smazání všech odpovědí a pak ke smazání samotné otázky.

2.2.4 Správa vytvořených formulářů

Správa vytvořených formulářů bude pravděpodobně další poměrně využívanou částí aplikace. Jednotlivým uživatelům umožní přehledné zobrazení již vytvořených formulářů, které lze různě třídit. O každém formuláři je možno si zobrazit jeho informace, přihlášené uživatele nebo přihlašovací údaje.

2.2.5 Přihlášení k formuláři

Přihlašování k formuláři bude pravděpodobně jedna z nejvíce používaných funkcí systému. Umožní jednotlivým uživatelům se přihlásit k danému formuláři. Každý formulář má vytvořen 36-místný hash klíč, díky kterému se lze přihlásit k formuláři.

Klíč je složen z náhodně vygenerovaných znaků spolu s kombinací aktuálního času a příklad hash klíče je zobrazen ve výpisu 1. Díky tomu nelze vygenerovat dva stejné hash klíče.

```
6a3K4e9K6q485N3i4x6O8a1o0q33889F3z19
```

Výpis 1: hash klíč

Pro přihlášení je možno využít buď integrovaný přihlašovací systém, nebo vložit vygenerovaný iframe na vlastní stránky. Při přihlašování jsou nejprve zobrazeny základní informace o formuláři (název, popis, datum). Pro samotné přihlášení k formuláři se musí uživatel nejprve přihlásit. To neplatí, pokud vyplňuje anonymní dotazník. Poté je mu umožněno odpovědět na vytvořené otázky. Nakonec je informován o úspěšném přihlášení k akci.

2.2.6 Správa chyb

Aplikace bude podporovat také jednoduchý a přehledný systém pro správu chybových hlášení. Tato sekce bude přístupná pouze administrátorům, díky čemuž bude možno opravit případné neodhalené chyby.

Jednotlivé chyby jsou uloženy v tabulce ErrorsMVC, která je v databázi. U jednotlivých chyb se eviduje datum, jméno controlleru, příslušné akce a také chybový výpis.

Také může nastat specifický problém, že z nějakého důvodu není k dispozici databáze. V tomto případě však nelze uložit chybovou zprávu do tabulky v databázi. Proto dojde nejprve k uložení chybového výstupy do XML souboru. Po přihlášení administrátora k systému a zobrazení stránky s chybovými stránkami dojde ke zkontrolování, zda-li jsou v XML souboru nějaké chyby. Pokud ano, dojde k jejich načtení a následnému uložení do databáze.

3 Model,View,Controller (MVC)

Model-View-Controller (MVC) [1, 2, 3] je důležitým architektonickým vzorem v počítačové technice již po mnoho let. Poprvé byl popsán v roce 1978 a byl pojmenován Thing-Model-View-Editor [13]. Později však byl tento název zjednodušen na Model-View-Controller.

Jedná se o výkonný, ale elegantní způsob oddělující byznys logiku od zobrazovací logiky. Aplikační rozdělení sice způsobuje menší množství problémů, ale celkové výhody převažují nad vynaloženým úsilím.

3.1 Historie MVC

Historie MVC sahá až do roku 1973, kdy byla popsána profesorem Trygvem Reenskaughem, tehdejším profesorem na „The Central Institute for Industrial Reaserch“.

Návrh vznikl při vymýšlení nového systému, který by měl vylepšit přijímání a odesílání zásilek v přístavu. Při pozorování samotného provozu si profesor všiml, že celý proces obsahuje velký počet na sobě nezávislých akcí a aktérů. Proto se rozhodl, že by bylo potřeba vytvořit systém, který by byl modulární a umožnil oddělení procesů a lidí.

Své poznatky poté rozdělil na několik základních částí:

- Možnost přestavby systému
- Existence modulů, které mohou být propojeny
- Oddělení jednotlivých systémů
- Škálovatelnost systému

3.2 Rozdělení architektury

MVC rozděluje aplikaci do tří hlavních (základních) komponent:

- Model- obsahuje sady tříd popisujících data, se kterými bude aplikace pracovat. Také obsahuje pravidla, jak může být s těmito daty nakládáno a manipulováno.

  Definované třídy velice často reprezentují data, která jsou načítána nebo uložena v databázi.

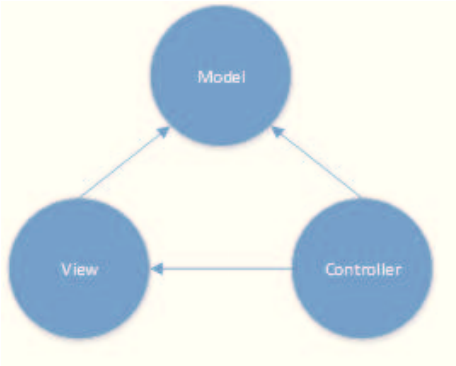
- View- pro koncového uživatele je na aplikaci nejdůležitější uživatelské rozhraní, se kterým pracuje. Můžeme mít elegantně navržen model, controller správně napsán a optimalizován, ale tyto věci uživatel nevidí a ani neocení. Celková interakce s uživatelem probíhá pouze přes View, které reprezentuje aplikaci a poskytuje první dojmy z aplikace.

  Samozřejmě pokud model a controller obsahuje chyby, tak ani sebelepší View nezajistí uživatelskou spokojenost. Na druhou stranu pokud vytvoříme špatně použitelné uživatelské rozhraní, které bude nepřehledné a chaotické, tak nám nepomůže, že model a controller fungují bez problémů.

View je zodpovědný za poskytování uživatelského rozhraní uživatelům. View obdrží data od příslušného controlleru, která následně zobrazí.

- Controller- Controllery v MVC jsou zodpovědné za zpracování uživatelských akcí. Provádějí úpravy Modelu v závislosti na uživatelském vstupu. Controllery jsou zaměřeny na aplikační tok, pracující s příchozími daty od uživatele a poskytující odchozí data příslušnému View.

Pro lepší představu je MVC ilustrován na obrázku 3.



Obrázek 3: Model view controller

3.3 MVC u webových aplikací

MVC byl nejprve určen pro lokální počítače, protože internet jak jej známe dnes, ještě neexistoval. Nyní je MVC používán u webových aplikací, které běží v nekonečné smyčce a čekají na uživatelskou aktivitu. Aplikace využívá klient-server komunikaci, a proto musí umět vytvořit uživatelský výstup pouze na základě url adresy a http hlaviček.

Díky tomu jsou zvýšeny nároky na controllery a rychlost celé aplikace. Po odeslání výstupu klientovi ukončí server spojení a poté čeká na další požadavky, které si nemůže předzpracovat.

3.4 MVC Frameworky

V současné době je k dispozici celá řada MVC frameworků, využívajících nejrůznější programovací jazyky. Mezi nejrozšířenější a nejpoužívanější patří například Ruby on Rails, Django, Java, ASP.NET MVC. Více informací se lze dočíst v referenci [1].

- **Ruby on Rails**

  Ruby on Rails je pravděpodobně jeden z nejpopulárnějších MVC frameworků pro vytváření Webu [1, 16].

  Funguje na základě dvou hlavních principů:

1. **Konvence nad konfigurací:** Tato myšlenka vychází z principu, že vývojář pracuje na vývoji webu několik let. Proto je dobré se shodnout, zda daná věc funguje a udělat z ní část frameworku.
2. **Neopakuj se, nebo to nech tak:** Centralizuje aplikační logiku a také kód. To by mělo umožnit napsání menšího množství kódu.

Jedna z největších výhod je programovací jazyk Ruby, který umožňuje dynamické programování a kompilaci za běhu.

- **Django a Python**

  Django sám sebe nazývá „framework pro perfekcionisty s termíny“ [1, 8]. Jedná se o webový Framework využívající jazyk Python [15]. Nejdříve byl navržen ke správě zpravodajských stránek společnosti „The World Company“. Později byl však uvolněn k volnému využívání.

  Django je dalším z celé řady MVC frameworků, avšak oproti ostatním nabízí možnost automatického vytvoření administrace projektu. Tato vlastnost je dynamicky vygenerována dle daného datového modelu. Mezi hlavní úkoly Djanga patří snadné vytváření rozsáhlých databází, které jsou řízeny webovými aplikacemi. Tento framework je zaměřen na propojitelnost a použitelnost komponent a také na rychlý vývoj s využitím konceptu „DRY“ („Don’t repeat yourself“ neboli „ neopakuj se“)

- **Java, Spring, Struts**

  Při využití Javy [1], jsou k dispozici tři hlavní MVC rozhraní:

  1. **Spring Framework-** Jedná se o Framework uvedený v roce 2003, umožňující vývoj J2EE aplikaci. Spring obsahuje komponentový kontejner a podporu transakcí integrovaných pomocí objektově orientovaných maperů. Zaměřuje je vytváření Java aplikací co nejjednodušeji a nejflexibilněji. Velmi často je využívám společně se Springem.
  2. **Apache Struts-** Tento Framework byl uveden v roce 2001 a byl vyvinut společností Apache. Struts je open-source Framework využívající JSP serveletů a JavaBeans. Později došlo ke spojení komunit WebWorks a Struts, které vytvořily Struts2.
  3. **JSF (Java Server Faces) -** Technologie byla vyvinuta společností Sun Microsystems a hlavní myšlenkou bylo umožnění přehlednějšího vytváření webových aplikací. Tento Framework poskytuje sadu standartních grafických komponent pro vývoj webových aplikací.

  Všechny tři frameworky sdílejí společný koncept model-view-controlleru, avšak se mírně liší v jejich životních cyklech.

- **PHP, Zend**

  Zend [1, 20] Framework, neboli ZF, je open-source Framework implementovaný v PHP, přináší trochu formality při vývoji PHP aplikací pro zachování flexibility, na kterou jsou PHP programátoři zvyklí.

- **MonoRail**

  Prvním úspěšným MVC framework pro ASP.NET byl MonoRail [1, 11], který je součástí projektu „Castle Project“. Ten byl založen Hamiltonem Verissimem de Olivera, který v roce 2006 založil firmu nabízející komerční podporu pro Castle project.

  MonoRail byl vytvořen kvůli nespokojenosti s ASP.NET WebForms a zahrnuje zjednodušení návrhu jednotlivých stránek. Jedná se o velmi flexibilní .NET aplikační framework podporující jednoduché využití nejrůznějších komponent. Podporuje velké množství rozšíření, ale i ve výchozím nastavení obsahuje komponenty jako NHibernate pro definování modelu, log4net pro přihlašování.

  MonoRail je často považován jako jeden z nejlepších open source projektů pro .NET, které jsou přikládány jeho zaměření, jednoduchosti a použitelnosti.

- **ASP.NET MVC**

  V únoru roku 2007 navrhnul Scott Guthrie („ScottGu“) z Microsoftu základy ASP.NET MVC [1, 2, 3, 4] při cestě letadlem na konferenci. Zprvu se jednalo o jednoduchou aplikaci mající pár set řádků kódu, avšak nabízený potenciál byl obrovský.

  Později na konferenci ALT.NET ukázal ScottGu skupině vývojářů „tuto super věc, kterou jsem vymyslel v letadle“ a zeptal se jich jestli vidí její přínos a co si o tom myslí. Jeho prezentace sklidila obrovský úspěch. V září 2007 byl navržen základní prototyp, kód a myšlenky.

  ASP.NET MVC vychází z mnoha základních segmentů jako jiné MVC platformy. Navíc nabízí výhody kompilace a úpravy kódu za využití programovacích jazyků VB a C#.

  ASP.NET MVC sdílí několik základních principů:

  – Neopakuj se
  – Konvence nad konfigurací
  – Modulovatelnost kdykoliv je to možné

  ASP.NET MVC poskytuje alternativu k ASP.NET Web Forms pro vytváření webových aplikací. ASP.NET MVC Framework je jednoduchý, dobře testovatelný s integrací ASP.NET prvků jako jsou master page, membership autentizací.

  MVC je standartní návrhový vzor, jenž je známý mnohým vývojářům. Některé webové aplikace z něj těží, jiné využívají tradiční návrhový vzor ASP.NET, založený na Web formech a postbacích. Jiné aplikace mohou kombinovat tyto 2 přístupy, kdy se navzájem nevylučují.

  Podpora pro Testování aplikace (TDD)

  ASP.NET MVC umožňuje jednodušší testování aplikace oproti WebForms aplikacím. Ve WebFormech je jedna třída využitá jak k zobrazení výsledku, tak i k reakci na uživatelský vstup.

Psaní automatizovaných testů pro aplikace založených na Web formech může být složité, jelikož k otestování jedné stránky musí být inicializovány všechny třídy, jejich child controls a ostatní prvky. Díky tomu je velice těžké napsat test, který se zaměří pouze na určitou část aplikace. Proto je testování aplikace v MVC jednodušší, jelikož vzájemně odděluje jednotlivé části, čímž usnadňuje testování individuálních komponent.

Výhody Asp.Net MVC aplikací:

- Jednodušší úprava díky rozdělení na model,view a controller.
- Nepoužívá view state nebo aplikací běžících na serveru. Díky tomu je ideální pro ty, kteří chtějí mít plnou kontrolu nad chováním aplikace.
- Lepší podpora testování (TDD).
- Výhodnější pro aplikace, které mají velký tým vývojářů a pro webové designéry, kteří potřebují velkou kontrolu nad chováním aplikace.

Funkce ASP.NET MVC:

- Oddělení aplikačních úloh (vstupní logika, business logika a UI logika)
- Rozšiřuje podporu routování, kdy se jedná o silnou komponentu umožňující mapování url adres.
- Podporuje již existující funkce ASP.NET

4 ASP.NET MVC3

ASP.NET MVC3 navazuje na předchozí verze ASP.NET MVC pro vytváření webových aplikací. Vychází z osvědčeného návrhového vzoru oddělujícího různé části aplikace pro lepší škálovatelnost, rozšiřitelnost a testovatelnost.

Mezi nejvýraznější změny patří uvedení nového zobrazovacího enginu Razor. Také byla vylepšena možnost při předávání dat mezi jednotlivými částmi aplikace.

4.1 Razor

Razor je nový zobrazovací engine určený pro vykreslování HTML stránek. Můžeme jej označit za první velkou úpravu zobrazování HTML od uvedení ASP.NET 1.0, který byl představen zhruba před 10 lety.

Ve starších verzích (MVC 1,2) byl jako výchozí zobrazovací engine použit WebForms View Engine, který je využíván v souborech ASPX/ASCX/MASTER a využíval syntaxe WebFormů. Tento způsob fungoval, ale byl navržen k vytváření stránek v grafickém editoru, proto se pro MVC moc nehodil.

Proto byl v ASP.NET MVC3 představen nový zobrazovací engine Razor, který byl navržen ke zjednodušení syntaxe ve stránkách ASP.NET MVC. Syntaxe Razoru byla navržena tak, aby byla jednodušší k psaní i čtení a neobsahovala „syntaktický šum“. To je dosaženo pomocí nového zápisu syntaxe, kdy se přestalo využívat podobné syntaxe jako v XML.

Při návrhu Razoru měli vývojáři tyto cíle:

- **Kompaktní, významný, plynulý:** Razor byl zaměřen k vytváření HTML stránek za využití minimální syntaxe. Nezaměřuje se na pouhé minimalizování počtu řádků kódu, ale také na způsob, jakým lze vyjádřit náš záměr.
- **Nevytvářet nový jazyk:** Razor využívá nové syntaxe, ve které dovoluje použít dosavadní znalosti programování v .NET.
- **Jednoduchý k naučení:** Jelikož je využíván již existující jazyk, jeho pochopení by mělo být velice jednoduché. Zjednodušeně řečeno: stačí psát HTML kód a pokud je vyžadován nějaký .NET kód, tak před něj dáme pouze @.
- **Práce v jakémkoliv editoru:** Jelikož je Razor zaměřen na HTML, je možno využít libovolný editor. Oproti tomu Visual Studio zvýrazňuje syntaxi a nabízí možnost využití IntelliSense pro ještě snadnější práci.
- **Vylepšení InteliSense:** Razor byl vymyšlen tak, že běžné IntelliSense bychom nevyužili, proto bylo prostředí vylepšeno a zobrazuje například dostupné vlastnosti jednotlivých objektů.

4.1.1 Syntaxe Razoru

Razor využívá kombinace HTML prvků a kódu programovacího jazyku C# nebo Visual Basic. K odlišení HTML a programovatelného kódu se využívá znak @, který je občas nazýván „magickým znakem".

Základní syntaxe

Znak @ můžeme využít jak pro jednořádkové, tak i pro víceřádkové příkazy. Jednoduchý příklad je uveden ve výpisu 2.

```
<p>
    Aktuální čas: @DateTime.Now
</p>

<p>
    @{
        var jmeno="Jan";
        var prijmeni="Novak";
     }
</p>
```

Výpis 2: Razor - základní příkazy

Podmínky, cykly

Oproti MVC2 je zjednodušená integrace podmínek a cyklů do zdrojového kódu stránky. Ve výpisu 3 je uveden jednoduchý příklad podmínky a cyklu foreach.

```
<p>
    @{
        if ( cislo  % 2 == 0) {
            <text>Číslo je sudé</text>
        }
        else {
            <text>Číslo je liché</text>
        }
     }

     @foreach( var item in Model )
       {
          @item.Name </br>
       }
</p>
```

Výpis 3: Razor - podmíky, cykly

Komentáře

Jednořádkové komentáře v Razoru začínají znakem „@*" a končí „*@". Pokud jsme uvnitř C# bloku lze využít „//" nebo „/* */"

Tabulkový prvek WebGrid

Jedná se o nový prvek v MVC3, který umožňuje velmi jednoduše zobrazit data do tabulek. Můžeme ho přirovnat ke GridView ve WebFormech. Ve starší verzi MVC nic podobného nebylo a musela se postupně vygenerovat potřebná data pomocí HTML tabulek.

Využití WebGridu je velmi jednoduché. Stačí pouze poslat z controlleru do View potřebná data, která uvedeme jako parametr WebGridu. Pokud chceme na stránce zobrazit více WebGridů s různými daty, je možno jako parametr použít třídu, která poskytne příslušná data.

```
@{
    WebGrid grid=new WebGrid(Model);
}
```

Výpis 4: Razor - WebGrid inicializace

Při vytváření nové instance můžeme WebGrid upravit pomocí parametrů. V tabulce 1 jsou uvedeny základní možnosti.

Název	Popis
source	Data k zobrazení
defaultSort	Specifikuje výchozí sloupec pro třízení
rowsPerPage	Počet řádků na stránku (defaultně je 10)
canPage	Umožní zapnout nebo vypnout stránkování
canSort	Umožní zapnout nebo vypnout třízení

Tabulka 1: WebGrid - základní parametry

Jakmile máme vytvořenou instanci webgridu, můžeme v libovolné části View WebGrid zobrazit.

```
@grid.GetHTML()
```

Výpis 5: Razor - WebGrid zobrazení na stránce

Zobrazení WebGridu na stránce můžeme upravit pomocí dodatečných parametrů[1] , které jsou uvedeny v tabulce 2.

tableStyle	CSS třída se styly
rowStyle	Třída se stylem pro liché řádky
alternatingRowStyle	Třída se stylem pro sudé řádky
Columns	Umožňuje definovat zobrazení sloupců

Tabulka 2: WebGrid - rozšiřující parametry

[1]Kompletní seznam parametrů je uveden na stránce http://msdn.microsoft.com/en-us/magazine/hh288075.aspx

4.1.2 Srovnání syntaxe WebForms a Razor

Mezi syntaxí WebForms a Razor byly provedeny úpravy, které vedly ke zjednodušení a zpřehlednění kódu.

Ve výpisu 6 je uvedeno několik příkladů, které srovnávají zdrojový kód napsaný v Razoru a WebFormech. Z těchto pžíkladů lze jasně vidět zjednodušení a zpřehlednění psaní zdrojového kódu v Razoru.

1. Kódový blok

```
Razor:
@{
        int a = 123;
        string b = "Ahoj";
 }

WebForms:
<%
    int a = 123;
    string b = "Ahoj";
%>
```

2. Kombinace textu a kódu

```
Razor:

Ahoj @jmeno , @prijmeni

WebForms:

Ahoj <%=jmeno %> , <%=prijmeni %>
```

3. Podmínky

```
Razor:

@if( podminka ) {
    <text>Plain Text></text>
}

WebForms:

<% if( podminka ) { %>
        Plain Text
<% } %>
```

Výpis 6: Razor - srovnání s WebForms

4.2 Předávání dat mezi stránkami v MVC3

Předávání dat mezi jednotlivými stránkami je důležitou součástí při vytváření každé aplikace. Mezi základní využívané prvky v ASP.NET WebForms patří session, viewstate, querystring a další.

ASP.NET MVC má ale odlišnou architekturu, proto se některé možnosti nedají použít a pro další případy bylo třeba definovat nové možnosti.

Samotné předávání dat lze rozdělit do dvou částí:

1. Předání dat z controlleru do View
2. Předávání dat mezi View

4.2.1 Předání dat mezi controllerem a View

Pokud chceme předat nějaká extra data z controlleru do View bez využití modelu, databáze nebo nějakých dalších prostředků, můžeme v MVC3 využít ViewData nebo ViewBag.

ViewData je možno označit jako datový slovník, kde přístup k datům je založen na principu „klíč/hodnota“ (např.: ViewData[“Jmeno”]=”Jan”).

ViewBag je prvkem využívajícím dynamických vlastností C# 4, které umožnuje elegantnější přidávání atributů jednotlivým prvkům. Základní konstrukce je deklarována ViewBag.Klic=“Hodnota“. ViewBag a ViewData jsou velice podobné prvky. ViewBag, na rozdíl od ViewData, nevyžaduje strongly-typed definici. Funkčně si jsou velice podobné a mohli bychom klidně říct, že jsou stejné.

V jednoduchém příkladu je vytvořen seznam obsahující tři zkratky řečí, poté je uložen do ViewBag.Langues a ještě jsou vytvořeny ViewBag proměnné obsahující aktuální čas a jméno. Pro splnění tohoto příkladu je třeba projít následující sérii kroků:

1. **Vytvoření dat v controlleru:** V controlleru máme vytvořený jednoduchý seznam, do kterého přidáme seznam zkratek několika zemí. Poté je uložíme do proměnné ViewBag.Languages. Dále vytvoříme další 2 ViewBag proměnné, kde v jedné je uložen aktuální čas a v druhé textový řetězec. Zdrojový kód controlleru je uveden ve výpisu 7.

```
public ActionResult About()
{
    List<string> Lang=new List<string>();
    Lang.Add(”CZ”);
    Lang.Add(”EN”);
    Lang.Add(”DE”);

    ViewBag.Languages = Lang;
    ViewBag.DateNow = DateTime.Now;
    ViewBag.Name = ”Petr”;

    return View();
}
```

Výpis 7: MVC - vytvoření controlleru

2. **Získání hodnot ve View:** Jakmile máme definována data v controlleru, můžeme je přidat do View. Zobrazení požadovaných dat je velice jednoduché a je ilustrováno ve výpisu 8.

```
@{
    <p>
        Jmenuji se: <b> @ViewBag.Name </b> <br/>

        Seznam zkratek zemí pomocí ViewBag: <br/><br/>

        @foreach ( var lan in ViewBag.languages )
        {
            @language <br/>
        }

        Aktuální čas: <b> @ViewBag.DateNow </b>

    </p>

}
```

Výpis 8: MVC - získání hodnot ve view

3. **Výstup na stránce:** Při spuštění aplikace a načtení příslušné stránky, dojde k zobrazení výstupu, který je na obrázku 4.

Obrázek 4: Výstup na stránce

4.2.2 Předávání dat mezi View

Při předávání dat mezi jednotlivými stránkami můžeme využít několika prvků, které jsou i v ASP.NET WebForms.

- QueryString
- Session
- Cookies

V ASP.NET MVC byla přidán nový prvek TempData, který bychom mohli přirovnat k ViewState v ASP.NET WebForms. Jedná se o proměnnou, která vydrží pouze jedno přesměrování mezi stránkami a poté zanikne.

Vytvoření a získání dat je stejné jako u ViewBag, data jsou vytvořena v controlleru a ve view dochází k jejich zobrazení.

```
Vytvoření TempData v controlleru:

TempData[”Jmeno”] = ”Jan”;


Získání hodnoty ve View:

@TempData[”Jmeno”]
```

Výpis 9: TempData - ukázka vytvoření

Pokud chceme, aby TempData vydržely více než jedno přesměrování (postback), je třeba využít metodu TempData.Keep(„Jméno“). Toto řešení je funkční, ale z praktického hlediska můžeme využít session.

5 Srovnání ASP.NET WebForms, ASP.NET MVC 2, MVC 3, MVC4

ASP.NET WebForms a MVC jsou dvě různé architektury vyvíjené společností Microsoft. Každá architektura má své silné i slabé stránky a jednotlivé frameworky mohou být použity jak samostatně, tak společně.

Před samotným vývojem aplikace by se měl každý vývojář zamyslet, kterou variantu je výhodnější použít, aby minimalizoval případné problémy.

5.1 ASP.NET WebForms

ASP.NET WebForms je jeden z části architektury ASP.NET. Jedná se o jeden z programovacích modelů, které mohou být využity k vytvoření ASP.NET webových aplikací.

WebForms jsou definovány jednotlivými stránkami, na které se dotazuje uživatel pomocí svého prohlížeče a obsahují jednotlivé prvky uživatelského rozhraní. To je tvořeno kombinací HTML, serverových ovládacích prvků a zdrojového kódu.

Využitím Visual Studia je možno vytvořit ASP.NET webovou aplikaci za využití výkonných IDE. Můžeme například jednoduše na stránku přetáhnout serverové ovládací prvky, rozvrhnout vzhled stránky a následně nastavit metody a události pro jednotlivé ovládací prvky. Pro samotný kód můžeme použít programovací jazyk Visual Basic nebo C#.

Mezi výhody WebForms patří:

- Podpora rychlého vývoje aplikace (RAD)
- Programování založené na událostech
- Velké množství ovládací prvků
- Výkonný data biding
- Podpora Ajaxu

Mezi nevýhody WebForms patří:

- Hůře čitelný zdrojový kód
- Složitější testování
- UI logika je společně s kódem

5.2 ASP.NET MVC

ASP.NET MVC je další z částí architektury ASP.NET. Využívá Model-View-Controller k oddělení jednotlivých částí. Samotný princip MVC byl popsán v kapitole 3.

Mezi výhody ASP.NET MVC patří:

- Poskytuje plnou kontrolu nad HTML
- Generuje čistější HTML kód
- Odděluje uživatelské rozhraní a aplikační logiku
- Lepší podpora testování
- Jednodušší integrace s jinými frameworky (JQuery, JSON)

Mezi nevýhody ASP.NET MVC patří:

- Slabá podpora ovládacích prvků
- Pomalejší vývoj aplikace
- Neobsahuje programování založené na událostech (pro někoho to může být výhoda)

5.2.1 ASP.NET MVC 2

V březnu 2010 vydal Microsoft novou verzi MVC frameworku, který navázal na první verzi. MVC2 obsahuje velké množství vylepšení zaměřených na zvýšení produktivity.

Mezi novinky patří:

- Možnost rozdělení projektů na menší celky (tzv. „Areas“).
- Možnost vykreslení sub-sekcí pomocí parametru Html.RenderAction
- Podpora asynchronních controllerů
- Možnost psaní „Strongly typed“ HTML prvků
- Množství nových funkcí, nástrojů a vylepšení API
- Vylepšení Visual Studia

5.2.2 ASP.NET MVC 3

MVC3 byl uveden necelý rok po předchozí verzi, proto nelze očekávat velké množství nových funkcí. Několik vylepšení se ale objevilo.

Mezi novinky patří:

- Nový zobrazovací engine Razor
- Podpora .NET 4
- Vylepšená validace modelu
- Vylepšena podpora JavaScriptu, jQuery Validace a JSON
- Podpora NuGet

Využití a validace Modelu v MVC3

V každé aplikaci je kontrola vstupů proměnných velice důležitou součástí, jelikož předpokládáme na vstupu data v nějakém formátu, ale uživatel může zadat něco jiného (typicky písmena místo číslic). V MVC byla vylepšena práce s modelem, která umožňuje definovat samotnou datovou strukturu, ale také omezení jednotlivých prvků. Lze definovat, jestli prvek musí být vyplněn, maximální délku nebo si definovat zcela vlastní pravidlo pro validaci.

Při vytváření aplikace je třeba definovat datový model. V tomto případě bude aplikace umožňovat jednoduché zaslání zpráv. Bude nutno zadat jméno, email a samotný text zprávy, kterou chceme odeslat. Základní model je zobrazen ve výpisu 10.

```
namespace TestovaciAplikace.Models
{
    public class KontaktniFormular
    {
        public String Jmeno { get; set; }
        public String Email { get; set; }
        public String Zprava { get; set; }
    }
}
```

Výpis 10: MVC - validace modelu 1

Tímto jsme definovali základní kostru modelu, která vypadá stejně, jako bychom definovali třídu ve WebFormech. MVC však umožnuje rozšířit tuto datovou strukturu a další atributy upřesňující význam jednotlivých položek. Tímto způsobem si můžeme ulehčit kontrolu položek při vložení na stránce.

Samozřejmě můžeme validaci provádět až po odeslání formuláře, ale tím bychom si jen přidali práci. Pokud bychom používali WebForms Engine, tak bychom museli na stránku přidat potřebné validátory a provést jejich nastavení. Toto řešení je sice funkční, ale má nevýhodu v tom, že znepřehlední kód stránky.

V MVC se tato validace provádí zároveň s definicí modelu. Tudíž je to mnohem přehlednější a jasnější. Upravený model je zobrazen na výpisu 11.

```
namespace TestovaciAplikace.Models
{
    public class KontaktniFormular
    {
        [Required( ErrorMessage = "Je třeba zadat jméno" )]
        [Display( Name = "Celé jméno")]
        [MaxLength(50, ErrorMessage = "Jméno může mít maximálně
            50 znaků")]
        public String Jmeno { get; set; }

        [Required( ErrorMessage = "Je třeba zadat emailovou adresu
            ")]
        [Display( Name = "Email")]
        [Email(ErrorMessage = "Email nemá správný formát")]
        public String Email { get; set; }

        [Required( ErrorMessage = "Zprávu je třeba vyplnit" )]
        [Display( Name = "Vaše zpráva")]
        [DataType(DataType.Multiline)]
        [MaxLength(1000, ErrorMessage = "Zpráva může mít
            maximálně 1000 znaků")]
        public String Zprava { get; set; }
    }
}
```

Výpis 11: MVC - validace modelu 2

Tímto jsme náš model poměrně rozšířili pomocí několika parametrů. Každý z parametrů nějak definuje určité vlastnosti. Parametry bychom mohli rozdělit do 2 skupin:

1. **Základní parametry-** tyto parametry umožní základní omezení jednotlivých prvků modelu. Je ale lepší je rozšířit pomocí dalších vlastností. Mezi typické základní parametry lze zařadit:

    - Required – určuje, jestli musí být položka vyplněna
    - DataType – rozšiřuje datový typ modelu (heslo,datum, telefon, atd.)
    - MaxLehgth – omezuje počet znaků, které lze zadat
    - Dále je možnost definovat vlastní validátory a další omezení

2. **Rozšiřující parametry-** upřesňují vlastnosti a chování základních parametrů. Většinou jsou využity při zobrazení stránky a výpisu různých chybových hlášení. Typickými představiteli jsou:

   - Name – pomocí tohoto parametru lze jednoduše zobrazit popis daného prvku
   - ErrorMessage – zobrazí chybové hlášení při špatném zadání

   Pokud využíváme vícejazyčné weby, je možno vložit odkaz z jazykového souboru přímo do parametru Name. Tímto bychom měli vytvořený model. Pro další využití je třeba vytvořit View.

Použití View v ASP.MVC3

V předchozím bodě jsme úspěšně definovali model stránky, který je třeba zobrazit uživateli. Náš View bude jednoduchý a bude zobrazovat jednotlivé položky a validátory.

Nejprve je třeba definovat, jaký model bude stránka využívat. Toho je dosaženo pomocí konstrukce ve výpisu 12.

```
@model TestovaciAplikace.Models.KontaktniFormular
```

Výpis 12: MVC - definice modelu ve View

Aplikace nyní ví, jaké může použít parametry a jaké mají atributy. Dále bude na stránku třeba umístit jednotlivé prvky modelu (jméno, e-mail a samotnou zprávu) a příslušný validátor. Také by bylo dobré před každý prvek vložit textové pole, ve kterém bude napsáno, co dané pole obsahuje. Zdrojový kód je zobrazen ve výpisu 13.

```
<div>
    @Html.LabelFor(x=>x.Jmeno)
</div>
<div>
    @Html.EditorFor(x=>x.Jmeno)
    @Html.ValidationMessageFor(x=>x.Jmeno)
</div>
```

Výpis 13: MVC - umístění prvků ve View

Na stránku jsme vložili tři různé prvky:

- LabelFor
- EditorFor
- ValidationMessageFor

Prvek LabelFor zobrazí hodnotu, která je uložena v parametru Display v modelu. V našem případě se jedná o tento řádek modelu, který je ve výpisu 14.

```
[Display( Name = "Celé jméno")]
```

Výpis 14: MVC - zobrazení hodnoty z Name

Toto řešení má výhodu, pokud data z modelu využíváme ve více částech aplikace. Nebudeme muset neustále psát popisky. Také je toto řešení vhodné pro vícejazyčné weby, jelikož lze do popisu vložit odkaz na příslušný Resource soubor. EditorFor vytvoří na stránce pole, kde uživatel napíše svá data. Posledním prvkem je ValidationMessageFor, který slouží jako validátor daného prvku.

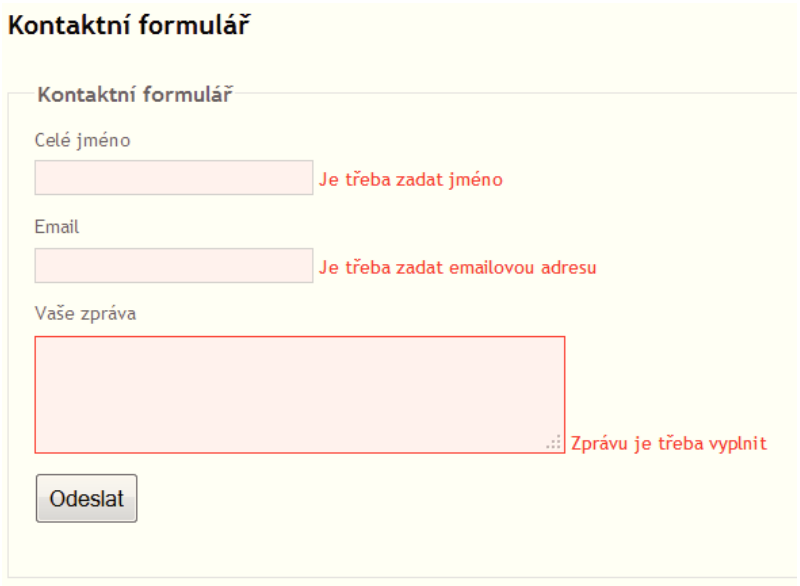
Při odeslání projde validátor všechny parametry prkvů na stránce a zjistí, zda jsou splněny. Pokud nejsou splněny, zobrazí u příslušného prvku chybové hlášení. Zdrojový kód našeho View je zobrazen na výpisu 15.

```
@model TestovaciAplikace.Models.KontaktniFormular
@{ViewBag.Title = "Poslání zprávy";}

<h2>Kontaktní formulář</h2>
@using (Html.BeginForm()){
    <fieldset>
        <legend>Kontaktní formulář</legend>

        <div>
            @Html.LabelFor(x=>x.Jmeno)
        </div>
        <div>
            @Html.EditorFor(x=>x.Jmeno)
            @Html.ValidationMessageFor(x=>x.Jmeno)
        </div>


        <div>
            @Html.LabelFor(x=>x.Email)
        </div>
        <div>
            @Html.EditorFor(x=>x.Email)
            @Html.ValidationMessageFor(x=>x.Email)
        </div>

        <div>
            @Html.LabelFor(x=>x.Zprava)
        </div>
        <div>
            @Html.EditorFor(x=>x.Zprava)
            @Html.ValidationMessageFor(x=>x.Zprava)
        </div>

        <p>
            <input type="submit" value="Odeslat" />
        </p>
    </fieldset>
```

Výpis 15: MVC - View

Celý kód vypadá přehledně a čitelně. Nejsou zde žádné zbytečné prvky nebo možnosti. Pokud bychom vytvořili stejnou stránku ve WebFormech, byla by mnohem nepřehlednější a taky mnohonásobně delší.

Po spuštění aplikace a pokusu o odeslání formuláře bude stránka vypadat jako na obrázku 5.



Obrázek 5: Výstup na stránce

Použití controlleru v ASP.MVC3

Již jsme vytvořili Model a View, ale ke správnému fungování potřebujeme vytvořit ještě controller, který zajišťuje celkovou komunikaci. Controllery v ASP.NET MVC reagují pouze na 2 metody a to GET a POST.

Při vytváření nového controlleru je důležitá konstrukce, která je na výpisu 16.

```
[HttpGet] //Nemusí být uvedeno
public ActionResult PosilaniZpravy()
{
    return View();
}
```

Výpis 16: MVC - definice controlleru1

Tento controller reaguje na metodu HttpGet a pomocí třídy ActionResult zobrazí uživateli daný view. Hodně controllerů vypadá stejně jako na výše uvedeném obrázku, kde každý controller musí obsahovat minimálně jeden „return View()“, kterým si zajistíme zobrazení příslušného view. V každém controlleru můžeme mít vice returnů, které mohou provést přesměrování na jiné stránky v aplikaci. Pro

Další controller reaguje na metodu POST, kdy uživatel stiskne tlačítko odeslat. Controller má v parametrech uveden námi definovaný model, který si přebere od View. Pokud je vše v pořádku a model je validní (jsou splněny všechny validátory), provede se samotné zpracování zaslané zprávy a poté pomoci metody RedirectToAction dojde k přesměrování na novou stránku. Pokud nějaké pole není vyplněno nebo je vyplněno chybně,

dojde k zobrazení daného formuláře s příslušnými chybovými zprávami. Zdrojový kód je zobrazen na výpisu 17.

```
[HttpPost]
public ActionResult PosilaniZpravy(KontaktniFormular model)
{
    if (ModelState.IsValid)
    {
        /* Zde je možnost uložení otázky do databáze */

        return RedirectToAction("NazevView","JmenoControlleru");
    }
    return View(model);
}
```

Výpis 17: MVC - definice controlleru2

Rozdíl mezi MVC2, MVC3
Mezi vydáním MVC2 a MVC3 uplynulo pouze 10 měsíců, tudíž nelze očekávat velké množství zásadních změn, ale pár se jich objevilo. Rozdíly jsou popsány v tabulce 3.

	MVC2	MVC3
Zobrazovací engine	WebForms view engine	Razor a WebForms view engine
Syntaxe	<% HTML_kód%>	@HTML_kód
Podpora JS,JQuery,JSON	Dobrá	Vylepšená
Podpora Layoutů	Pouze master page(.master)	Master page,LayoutPage
Tabulkové prvky	Ne	Ano
Dependency injection	Ne	Ano
Předávání dat mezi stránkami	TempData, ViewData	TempData, ViewData, ViewBag

Tabulka 3: Srovnání MVC2, MVC3

Nejdůležitější a zásadní změnou bylo uvedení nového zobrazovacího enginu Razor, který je popsán v kapitole 4.1.

5.2.3 ASP.NET MVC 4

V průběhu vypracovávání diplomové práce byla uvedena nová verze ASP.NET MVC s pořadovým číslem 4.

ASP.NET MVC4 je framework sloužící k vytváření webový aplikací za využití osvědčeného návrhového vzoru a ASP.NET MVC. MVC4 se zaměřuje a vylepšuje vývoj aplikací určených pro mobilní zařízení.

Při vytváření nové aplikace je možno využít nový typ projektu přímo určený pro mobilní zařízení. MVC4 také integruje jQuery pro mobilní zařízení, jež bychom mohli definovat jako rozhraní založené na HTML5 kompatibilní se všemi platformami (windows phone, android, iPhone). MVC4 umožňuje také definování obsahu dle konkrétního zařízení. Oproti verzi MVC3 nabízí opět řadu vylepšení, hlavně v oblasti vývoje pro mobilní zařízení:

- **ASP.NET Web API-** Jedná se o nová API umožňující vytváření http služeb pro velký počet klientů využívajících jak klasické prohlížeče, tak mobilní zařízení.
- **Nové Project Templaty-** Stávající výchozí templaty byly nahrazeny novými a modernějšími. Hlavní výhodou nových templatů je využití techniky, která se automaticky přizpůsobí prohlížeči, čímž aplikace vypadá dobře jak na desktopu, tak mobilním zařízení.
- **Mobilní Project Templaty-** Tyto templaty umožňují snadnější vytváření aplikací pro mobilní zařízení. Jsou založeny na mobilním jQuery a knihovnách podporujících dotykové ovládání.
- **Zobrazovací módy-** Umožní aplikaci vybrat požadovaný View podle prohlížeče, který o něj požádal. Jinak bude vypadat View pro desktopovou aplikaci a jinak pro mobilní zařízení.

6 Srovnání vývoje aplikace MVC a WebForms

Vývoj aplikace pomocí MVC a Webforms využívá stejný programovací jazyk (C# nebo VisualBasic), ale vývoj samotné aplikace je rozdílný.

Obě architektury nabízejí plusy a mínusy, a proto je se třeba před začátkem vývoje rozhodnout, která architektura bude „lepší " volbou k řešení daného problému.

Při výběru architektury bychom měli zvážit následující základní možnosti:

1. **Rychlý aplikační vývoj (RAD)–** Pokud chceme vyvíjet aplikaci rychle, je správnou volbou WebForms. Při rychlém aplikačním vývoji se předpokládá, že aplikace nebude vyžadovat značnou údržbu a je ji možno vytvořit za pomocí více programátorů v krátkém čase. Také se předpokládá, že aplikaci je třeba vytvořit co nejlevněji nebo za krátkou dobu.

2. **Automatické testování-** Pokud vytváříme aplikaci, kterou je třeba důkladně otestovat, je správnou volbou MVC.

3. **Znalost technologie-** Pokud známe pouze jednu technologii, nemá cenu aplikaci vyvíjet v té druhé. Toto samozřejmě neplatí, pokud jsou vyžadovány nějaké speciální důvody nebo požadavky.

4. **Využívané prvky-** Jestli budeme v aplikaci využívat převážně tabulkové prvky (gridview, repeater, atd.), je lepší využít WebForms, který jich nabízí velké množství. Výhoda využití Webforms spočívá ve snadném propojení více prvků ná stránce pomocí parametrů.

5. **Kontrola nad výstupem-** Pokud požadujeme dobře čitelný a přehledný HTML výstup, je lepší využít MVC.

6. **Interakce s jinými technologiemi-** Při využití Ajaxu, JQuery, JSON a dalších technologií v aplikaci je lepší využít MVC, jelikož je lépe optimalizován.

6.1 Životní cyklus stránek v MVC a WebForms

Jednotlivé architektury lze dobře porovnat také pomocí životního cyklu stránky, který určuje chování aplikace.

Životní cyklus stránky ASP.NET MVC bychom mohli rozdělit do následujících pěti kroků:

1. **Inicializace-** Tento krok se provádí při prvním požadavku na aplikace. Nejprve jsou spuštěny metody Application_Start() umístěné v souboru Global.asax. Tato metoda obsahuje pravidla, která jsou využívána pro výběr controlleru.

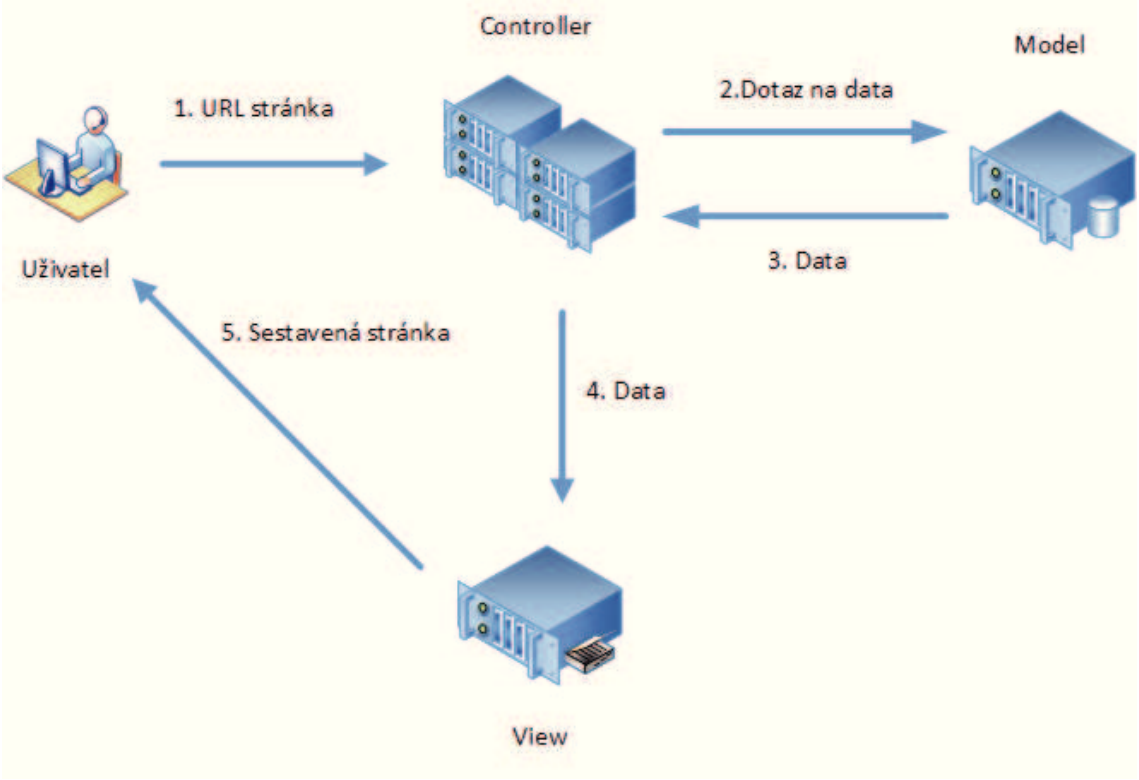
2. **Routování-** Při routování dochází k porovnání příchozí url adresy s routovacími pravidly. Podle toho je následně vybrán příslušný controller, jemuž jsou předány další parametry, které jsou volitelné a jsou v url adrese.

3. **Vybrání controlleru-** Jakmile je vybrán správný controller, je třeba určit přesnou akci v daném controlleru.

4. **Provedení akce v controlleru-** Po vybrání správné akce je třeba naplnit daný model příslušnými daty a provést další požadované úkoly. Nakonec dojde k zavolání příslušného view.

5. **Zobrazení view-** Příslušný view obdrží nejdříve od view data, které pak zpracuje. Následně dojde k vygenerování samotné stránky, která je zobrazena uživateli.

Princip životního cyklu stránky je ilustrován na obrázku 6.



Obrázek 6: Životní cyklus MVC stránky

6.2 Životní cyklus WebForms stránky

Životní cyklus ASP.NET WebForms stránky je zcela odlišný než u MVC. Při vyžádání stránky uživatelem dojde ke spuštění série událostí, které postupně procházející celý životní cyklus stránky.

Každá ASP.NET stránka začíná vytvořením požadavku klienta ke zobrazení stránky. Server nahraje danou stránku, kterou zkompiluje a provede na straně serveru. Následně

jsou vygenerovány HTML značky, které je možné zobrazit v prohlížeči. Po zpracování a zobrazení stránky dojde k jejímu odstranění ze zpracování.

1. **PreInt-** Tuto událost bychom mohli označit za vstupní bod životního cyklu ASP.NET WebForms stránky. Jedná se o jakousi předinicializaci, takže je možnost přistupovat ke stránce ještě předtím, než je vytvořena. Můžeme zde přistupovat k masterPage, tématům a vytvářet ovládací prvky.

2. **Init-** Tato událost je spuštěna po inicializaci všech prvků na stránce a vybrání témat. Je zde možno taky pracovat s událostmi jednotlivých komponent.

3. **PreLoad-** Tato událost je spuštěna po načtení dat z ViewState a PostBack do jednotlivých komponent.

4. **Load-** Zde můžeme nastavovat a číst hodnoty komponent. Nastavovat parametry pro spojení s databázemi.

5. **PreRender-** Jedná se o poslední možnost, kdy lze změnit vlastnosti komponent a upravit již načtenou stránku. Jakmile je tato událost provedena, dojde k uložení pozměněných vlastností jednotlivých komponent do ViewState.

6. **Render-** Dojde k vygenerování dané stránky do HTML.

7. **Unload-** Poslední krok v životním cyklu stránky. Dojde k uzavření spojení, uvolnění zdrojů, atd.

Jelikož byla aplikace vyvíjena při využití obou architektur, je možné provést vzájemné srovnání, na které jsme narazili při vývoji.

Vytváření aplikace za využití WebForms nebylo ze začátku nijak složité. Zjednodušeně by se dalo říct, že stačilo spustit Visual Studio a začít programovat.

Při vytváření MVC aplikace je nutnost začít uvažovat mírně odlišně než při vývoji WebForms aplikace. We WebFormech stačí vytvořit novou stránku (.aspx) a hned ji lze zobrazit. V MVC toto nefunguje a je třeba vytvořit jak stránku, tak i příslušný controller. Na tento problém si však lze velice rychle zvyknout.

Před začátkem implementace systému pomocí MVC3 bylo nutné nastudování a pochopení základního principu a vývoje. To mělo za důvod počáteční zpomalení vývoje aplikace.

Jakmile jsem danou problematiku pochopil a nastudoval, nebyl základní vývoj v MVC3 moc odlišný oproti WebForms. Při samotné implementaci se však objevilo množství menších i větších problémů, které bylo nutno vyřešit.

Při implemetaci systému v MVC i WebForms nastaly dva hlavní problémy. Jedním byla implementace autorizací a druhým vytvoření dynamicky generovaných otázek. Problém s autorizacemi spočíval v tom, že systém měl podporovat různé autorizační systémy. Největší problém byl s Windows Live. Jelikož nejprve nepodporovali autorizaci pomocí OAuth protokolu ve verzi 2, tak se mi ji dlouho nedařilo úspěšně implementovat.

Dalším problémem bylo vytvoření dynamicky generovaných otázek. Tento problém byl v MVC vyřešen za vytvoření JavaScriptu. Nejprve bylo nutné umožnit uživateli vygenerování libovolné struktury otázek. Dalším problém nastal při odeslání dané stránky a následném rozpoznání jednotlivých otázek v controlleru a uložení do databáze.

7 Přihlašování

Uživatelská autorizace je další důležitou částí aplikace. Umožňuje jednotlivým uživatelů přiřadit nejrůznější oprávnění, díky kterým mohou mít v rámci aplikace jiné možnosti a funkce.

Náš systém podporuje různé formy uživatelské autorizace. Je k dispozici integrovaný autorizační systém nebo lze využít jednu z následných autorizačních služeb:

- VŠB LDAP- je popsán v kapitole 7.2
- Windows LIVE-je popsán v kapitole 7.3
- Facebook- je popsán v kapitole 7.4

Výše zmíněné autorizační služby mají pro uživatele výhodu v tom, že se nemusí znovu registrovat, ale stačí využít stávající přihlašovací údaje. V zadání práce byla zmíněna implementace autorizace pomocí portálu Netstudent.cz. Ten jsem po dohodě s vedoucím práce neimplementoval a místo něj jsme využili autorizace pomocí Facebooku.

7.1 Integrovaný autorizační systém

Aplikace podporuje vlastní autorizační systém umožňující registraci jakéhokoliv uživatele. Samotná registrace není nijak náročná. Stačí pouze vyplnit registrační formulář a poté je možné aplikace bez jakýchkoliv problémů používat.

Tento způsob vyžaduje mít další přihlašovací jméno k nové službě, proto je pro mnohé výhodnější využít nějaký z podporovaných autorizačních služeb za využití již existujícího účtu.

7.2 VŠB LDAP

Přihlašování pomocí VŠB LDAP je dostupné všem studentům a zaměstnancům VŠB-TU Ostrava. K přihlášení jim stačí pouze využít své přihlašovací jméno a heslo, které používají k přihlášení a k dalším školním službám. V současné době není v naší aplikaci používán, protože neumožňuje bezpečné přihlašování přes SSO. Naše žádost o využívání SSO byla zamítnuta, jelikož ji nelze využít ve studentských aplikacích.

Z cvičných důvodů bylo naimplementováno méně bezpečné řešení, které umožní uživateli využít školní přihlašovací údaje. Toto řešení obsahuje bohužel velký bezpečnostní problém, jelikož uživatel sdělí své přihlašovací jména a heslo naší aplikaci, ta je následně ověří u školního autorizačního serveru, jestli jsou pravdivé. Tuto autorizaci nepoužíváme, jelikož zadané přihlašovací údaje by mohly být velice snadno zaznamenávány a následně zneužívány.

Autorizace pomocí LDAP využívá vlastnosti ze třídy System.DirectoryServices.Protocols. Samotné spojení s autorizačním serverem využívá zabezpečené připojení přes SSL. Při autorizaci pomocí LDAP je nutno nejprve vytvořit novou instanci třídy LdapConnection,

která má v parametru instanci třídy LdapDirectoryIdentifier a obsahuje url adresu i číslo portu.

V rámci VŠB-TU je instance třídy LdapConnection definována na výpisu 18.

```
LdapConnection con = new LdapConnection(new LdapDirectoryIdentifier("
    ldap.vsb.cz",636))
```

Výpis 18: LDAP- inicializace LdapConnection

Následně stačí nastavit šifrované spojení SSL a autorizační typ. Poté je možné otevřít spojení a odeslat požadavek, ve kterém u autorizačního serveru zjistíme, zda je daný login platný. Nasatavení je zobrazeno na výpisu 19.

```
con.SessionOptions.SecureSocketLayer = true;
con.AuthType = AuthType.Anonymous;
con.Bind();

SearchRequest request = new SearchRequest("",String.Format("(&(objectClass=Person)(
    uid={0}))",username), SearchScope.Subtree);
SearchResponse response = (SearchResponse)con.SendRequest(request);
```

Výpis 19: LDAP - nastavení připojení

Pokud zjistíme, že daný login existuje, můžeme se zkusit připojit na server s využitím zadaného uživatelského jména a hesla. Jestliže vše proběhlo v pořádku, můžeme si z response zjistit další požadované údaje: mail, login, jméno, příjmení.

Celý autorizační kód pomocí LDAP je v příloze B.

7.3 Windows Live

Další možností uživatelské autorizace je využití Windows Live. V současné době jej využívá okolo 500 miliónů uživatelů, díky tomu je možno zpřístupnit naši aplikaci velkému množství uživatelů.

Microsoft nenabízí pouze autorizaci uživatele pomocí Live účtu, ale poskytuje také celou řadu rozhraní API, které umožňují práci s dalšími službami jako je Hotmail, Windows Live Messenger, Microsoft Skydrive a dalšími podporujícími Live Connect.

Při práci lze využít následující API:

- Identity API- je možné nazvat jako základní API, které umožnuje provádět dvě základní operace:
    1. **Personalizace-** umožnuje uživateli přihlášení a zaregistrování k dané stránce
    2. **Uživatelský profil-** umožňuje získání informačních dat o uživateli, jako je například jméno, příjmení,..

- Hotmail API- umožňuje vytvoření a přístup k uživatelským kontaktům a kalendářům v Hotmailu. Hotmail API lze rozdělit na několik základních částí:
  - Vytváření a čtení kontaktů
  - Pracovat s kalendáři
  - Vyhledávání lidí v kontaktech
- Skydrive API- umožnuje pracovat s datovým úložištem SkyDrive. Je možno vytvářet, upravovat a přistupovat k dokumentům, fotkám a dalším souborům, které jsou uloženy ve SkyDrive.
- Messenger API- umožňuje implementaci reail-time chatovacího programu s využitím protokolu XMPP.

Autorizace uživatele pomoci Windows Live:
Pokud chceme na stránkách využívat autorizaci pomocí Windows Live, tak musíme provést řadů kroků, které je nutno splnit:

1. **Mít zaregistrován uživatelský učet**
   Základním požadavkem je mít funkční emailový učet, který končí na:
   - @hotmail.cz
   - @hotmail.com
   - @live.com
   - @windowslive.com

   Bez splnění této podmínky nemůžeme pokračovat.
2. **Registrace nové aplikace**
   Pro samotnou autorizaci musíme mít zaregistrovanou aplikaci u Microsoftu. Tu lze zaregistrovat na stránkách společnosti Microsoft[2]. Nejprve se musíme přihlásit pomocí Live účtu a poté je možné záložce „My Apps“ vytvořit novou aplikaci.

   Při vytváření aplikace je třeba zadat její název a poté url adresu, ze které se k této aplikaci můžeme přihlásit. Tímto jsme úspěšně zaregistrovali novou aplikaci a v naší aplikaci se na ni budeme odkazovat pomocí Client ID, které je uvedeno v nastavení aplikace.

[2] http://msdn.microsoft.com/en-us/live

3. **Autorizace uživatele**
   Nyní můžeme přistoupit k naprogramovaní a autorizaci v naší aplikaci. Pro samotnou autorizaci můžeme využít jednu z těchto kategorií:

   - JavaScript
   - C#
   - Objective-C
   - Java
   - REST

V naší aplikaci budeme využívat autorizace uživatele pomocí JavaScriptu. Při využití JavaScriptovské autorizace je nutno na stránku přidat soubor „wl.js“ , který vložíme na stránku pomocí kódu ve výpisu 20.

```
<script src="https :// js . live .net/v5.0/wl. js " type="text/ javascript "></script>
```

Výpis 20: Windows Live - přidání JS

Pro autorizaci je třeba nejdříve inicializovat funkci WL.Init, která si načte z JavaScriptovské knihovny podporované funkce, které je možno následně využívat.

Funkce WL.ini musí být volána vždy před ostatními funkcemi, jinak by nemusely další funkce fungovat správně nebo taky vůbec.

V našem případě je funkce Wl.Init definována ve výpisu 21.

```
WL.init({  client_id :  "0000000440A8D15",
                redirect_uri :  "http :// formsmvc.cs.vsb.cz/Account/LogOn",
               response_type: "token"
          });
```

Výpis 21: Windows Live - inicializace

Jakmile jsme inicializovali funkci Wl.Init(), můžeme se pokusit získat uživatelská data. K tomu slouží funkce WL.getSession(), která se pokusí získat ze serveru session, ve kterém je umístěn access_token (přístupový kód). Jestliže je přístupový kód platný, můžeme následně získat informace o daném uživateli. Ten je možno získat s využitím kódu zobrazeného ve výpisu 22.

```
var session = WL.getSession();

if (session)
{
    displayGreetigns(session); // třída , které  zjistí  uživatelká  data
}
else
{
    WL.ui({ name: "signin",  element: "signInButton" });
}
```

Výpis 22: Windows Live - zobrazení informací

Jestliže není obsah session prázdný, můžeme s pomocí funkce WL.API získat požadované informace o uživateli a uložit si je do proměnných, které je možné dále využívat. Pokud je obsah session nulový, musíme zobrazit na stránce přihlašovací tlačítko pomocí funkce WL.ui(). Další funkce WL.login s parametrem „scope“ určuje, jaké informace můžeme o uživateli získat. V našem případě je možno zjistit základní informace o uživateli (jméno, příjmení, email).

Pokud bychom ale využili dalších parametrů příkazu scope, tak bychom mohli zjistit mnohé další informace (datum narození, pohlaví, atd.)

Jelikož Microsoft oznámil zrušení Windows Messengeru a jeho následný převod na službu Skype, je možné, že tato autorizace nebude fungovat korektně. Převod by měl proběhnout na přelomu dubna a května 2013, proto je třeba zkontrolovat funkčnost a případně ji upravit podle nových podmínek.

Kompletní zdrojový kód pro autorizaci pomocí Windows Live je uveden v příloze C.

7.4 Facebook autorizace

Další možnosti uživatelské autorizace je využití Facebooku. V dnešní době využívá Facebook kolem jedné miliardy uživatelů, proto je využití autorizace pomocí tohoto serveru velkou výhodou. Facebook využívá také protokol OAuth a nabízí k dispozici SDK pro snazší integraci.

Pro implementaci autorizace jsou momentálně k dispozici dvě oficiální SDK a to pomocí:

- JavaScriptu
- PHP

Také je možno využít několik dalších SDK, které sice nejsou oficiální, ale měly by nabízet stejnou funkčnost. Neoficiální řešení podporují tyto programovací jazyky:

- Python
- .NET (C#)
- Java (Spring, BlackBerry)
- Ruby
- Flash (ActionScript)

V naší aplikaci budeme využívat JavaScriptu, který je pro implementaci velmi jednoduchý. Facebookové SDK pro Javascript nabízí množství funkcí, které umožňují využít přihlašování pomocí Facebookového loginu a interakci s dalšími Facebookovskými API.

Ty obsahují Graph API zobrazující dialogy. Dále umožňuje zobrazení Social Pluginů, které zajišťují propojení s Facebookovskou stránkou.

Mezi nejpoužívanější Social Pluginy patří:

- **Login Button** slouží k přihlášení uživatelů k dané stránce
- **Like Button** dovolí uživateli zobrazit stránku na jeho FB profilu
- **Coments** umožnuje komentování obsahu stránky.

Pokud chceme využívat autorizace pomocí Facebooku, musíme splnit následující kroky:

1. Zaregistrovat aplikaci
2. Vytvořit přihlašovací JavaScript
3. Přidat na stránku přihlašovací tlačítko
4. Získat uživatelské informace a integrovat uživatele do systému

7.4.1 Registrace nové FB aplikace

Abychom mohli zaregistrovat novou aplikaci, musíme mít funkční FB účet. Poté je možno na stránce https://developers.facebook.com/apps vytvořit novou aplikaci. Při vytváření je nejprve nutno zadat název a jméno nové aplikace.

Při registraci je také nutno zadat doménu, ze které se budou uživatelé přihlašovat a také vybrat způsob, jakým se bude aplikace integrovat s Facebookem. V našem případě bude nutné zatrhnout možnost „Přihlášení pomocí Facebooku“ a doplnit url adresu, na které bude umístěn přihlašovací formulář.

Tímto bychom měli zaregistrovanou aplikaci a můžeme přistoupit k dalším krokům. Pro samotnou autorizaci budeme potřebovat číslo aplikace, které je vidět v nastavení. Tímto je aplikace připravená k uživatelskému přihlašování.

Samotná autorizace na webové stránce funguje následovně:

1. Uživatel klikne na facebookovské tlačítko „Přihlásit se“
2. Dojde k zobrazení vyskakovacího okna
3. Uživatel zadá své jméno a heslo
4. Pokud je vše v pořádku, je uživatel přihlášen, v opačném případě musí postup zopakovat

7.4.2 Vytvoření přihlašovacího Javascriptu

Poté, co máme zaregistrovanou aplikaci, můžeme přistoupit k samotné uživatelské autorizaci.

Nejprve je třeba na stránku vložit JavaScriptový odkaz, který umožní následující práci. Požadovaný JS přidáme na stránku pomocí kódu uvedeného ve výpisu 23:

```
<script type="text/javascript" src="http://connect.facebook.net/en_US/all.js"></script>
```

Výpis 23: Facebook - načtení JS

Po načtení autorizačního JavaScriptu je nutno inicializovat funkci FB.Init(), která umožní použití dalších požadovaných FB funkcí.

V každé aplikaci by měla být funkce FB.Init() na prvním místě, jelikož zajišťuje prvotní inicializaci autorizačního procesu. Pokud bychom měli před funkcí FB.Init() nějaké jiné facebookovské funkce, mohla by být omezena jejich správná funkčnost.

Funkce FB.Init() je pro naší aplikaci zobrazena ve výpisu 24

```
FB.init({ appId: '200418406669174', // cislo aplikace
    status: true, // umozni aktualizaci dat
    cookie: true, // podpora cookie
    xfbml: true // umoznuje parsovat XFBML tagy
        });
```

Výpis 24: Facebook - inicializace

Funkce FB.Init má několik parametrů. Nejdůležitějším je appID, který definuje číslo naší aplikace. Dále jsou využity parametry status, cookie a xfbml, které nejsou povinné.

Jelikož již máme inicializovánu funkci FB.Init(), můžeme postoupit k samotnému přihlášení. To se provádí pomocí funkce FB.Login. Po přihlášení uživatele budeme chtít zjistit jeho jméno a email, takže je třeba se přihlásit na událost auth.login, která je automaticky spuštěna poté, co se uživatel přihlásí.

Funkce pro přihlášení uživatele jsou zobrazeny na výpisu 25

```
    FB.login(function(response) {
});

FB.Event.subscribe('auth.login', function (response) {
    FB.api('/me', function (response) { });

    UserInfo(); //volani funkce ktera zjisti uzivatelske informace
});
```

Výpis 25: Facebook - přihlášení

Při odhlašování je postup velmi podobný. Nejprve je inicializována funkce FB.Logout() a poté událost na danou funkci, která opraví uživatelské údaje. Odhlášení uživatele je zobrazeno na výpisu 26

```
  FB.logout(function (response) {});

FB.Event.subscribe('auth.logout', function (response) {
    FB.api('/me', function (response) { });
    UserInfo(); //volani funkce ktera zjisti uzivatelske informace
});
```

Výpis 26: Facebook - odlášení

Funkce UserInfo() se pokouší zjistit jméno a emailovou adresu uživatele. Uživatelská data jsou zobrazena ve výpisu [27]

```
function UserInfo() {
  FB.api('/me', function (response) {
      var query = FB.Data.query('select name,email, pic_square from user where uid={0}',
          response.id);
      query.wait(function (rows) {
          name = rows[0].name;
          mail = rows[0].email;
      });
  });
}
```

Výpis 27: Facebook - získání dat

Tímto je celý autorizační JavaScript pro přihlášení hotov.

Posledním krokem je přidání tlačítka, které umožní samotné přihlášení nebo odhlášení. Přihlašovací tlačítko vložíme na stránku pomocí kódu ve výpisu 28

```
<fb:login-button autologoutlink="true" perms="email"></fb:login-button></td>
```

Výpis 28: Facebook - přihlašovací tlačítko

Parametr autologoutlink umožní uživateli se odhlásit. Další parametr perms definuje informace, které je možno o uživateli získat.

Kompletní zdrojový kód pro autorizaci pomocí Facebooku je uveden v příloze D.

7.5 OAuth

OAuth [14] je otevřený standard pro autorizaci, poskytující metody umožňující uživatelům přístup ke zdrojům na serveru. Také poskytuje koncovým uživatelům možnost, jak autorizovat služby třetích stran bez nutnosti sdělovat jim své přihlašovací údaje. OAuth podporuje v dnešní době množství serverů, kdy mezi nejznámější patří: Dropbox, Facebook, Google, Microsoft, PayPal, Twitter, Yahoo!.

7.5.1 Historie

OAuth byl poprvé popsán v listopadu 2006 Blainem Cookem při vývoji OpenID autorizace pro Twitter. Současně Gnolia (Ma.Gnolia) potřebovala řešení, které by umožnilo jejím uživatelům autorizace jejich widgetů. Proto se sešli se zástupci Twitteru k diskuzi o možnosti využití OpenID pro autorizaci uživatelů Twitteru a Magnolie.

Poté byla v dubnu 2007 sestavena malá skupina lidí, která měla za úkol navrhnout předlohu nového otevřeného projektu. V červenci 2007 byly vypracovány základní specifikace a již v říjnu byl spuštěn OAuth 1.0. V současné době je k dispozici protokol ve verzi 2.0, který ale není zpětně kompatibilní s první verzí. OAuth2 je zaměřen na jednoduchou implementaci na straně uživatele, podporující specifické autorizační toky pro webové, mobilní a další aplikace.

7.5.2 Princip fungování

OAuth využívá pro autorizaci Tokenů generovaných poskytovateli služby, místo uživatelských údajů pomocí chráněných zdrojů. Celý proces využívá dvou typů Tokenů:

- **Request Token-** Aplikace vyzve uživatele, aby se autorizoval k přístupu ke zdrojům. Uživatelem autorizovaný v Reqeuest Token je poté vyměněn za Acces Token, který musí být použit pouze jednou a nesmí být použit pro žádný jiný účel. Je doporučeno, aby Request Token měl omezenou dobu platnosti.
- **Access Token-** Tento token využívá aplikace k přístupu k uživatelským údajům za využití uživatelského jména. Access Tokeny mohou limitovat, jaká data může aplikace získat.

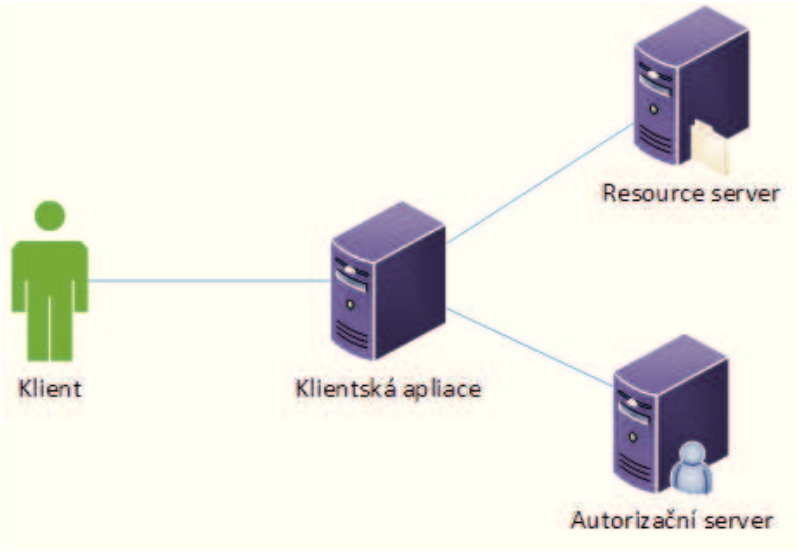
Autorizace pomoci protokolu OAuth lze rozdělit na 4 skupiny:

1. Uživatel
2. Klientská aplikace
3. Resource server - server, na kterém jsou uloženy uživatelská data
4. Autorizační server - jedná se o server, který umožňuje uživatelskou autorizaci

Základní princip autorizace bychom mohli popsat následovně:

1. Uživatel se připojí na stránku
2. Autorizuje se pomocí vybraného serveru
3. Aplikace obdrží uživatelská data
4. Uživatel je autorizován

Základní rozdělení je zobrazen na obrázku 7.



Obrázek 7: OAuth - základní princip fungování

V detailnějším rozboru probíhá autorizace následovně:

1. Uživatel se připojí na webovou aplikaci
2. Klientská aplikace podporuje OAuth protokol pro připojení k danému serveru
3. Uživatel si vybere autorizační server, poté je přesměrován na stránku, kde zadá své přihlašovací údaje.
4. Autorizační server ověří uživatelská data a provede přesměrování ke klientovi s autorizačním kódem.
5. Uživatel je přesměrován na klientskou aplikaci s ověřovacím kódem.
6. Klientská aplikace odešle ověřovací kód spolu s Client ID a Client secret, které obdržel od Autorizačního serveru.
7. Autorizační server ověří autorizační kód, Client ID a Client Secret a vrátí access token.
8. Poté je uživatel autorizován.

Pro lepší představu je autorizace zobrazena v příloze E.

8 Testování zátěže na přístup většího počtu uživatelů

Poslední částí diplomové práce je otestování systému při připojení většího počtu uživatelů. Tyto testy nám pomohou odstranit případné chyby, které by mohly nastat. Také umožní optimalizaci zdrojového kódu pro rychlejší načítání stránek a menší zatížení serveru.

Pro testování na lokálním počítači je nutno mít v systému nainstalovanou internetovou a informační službu (IIS). Jednotlivé testy je možno vytvářet přímo ve Visual Studiu 2010 nebo 2012. Je však třeba mít nainstalovanou verzi Ultimate, jelikož ostatní verze neobsahují všechny druhy testů.

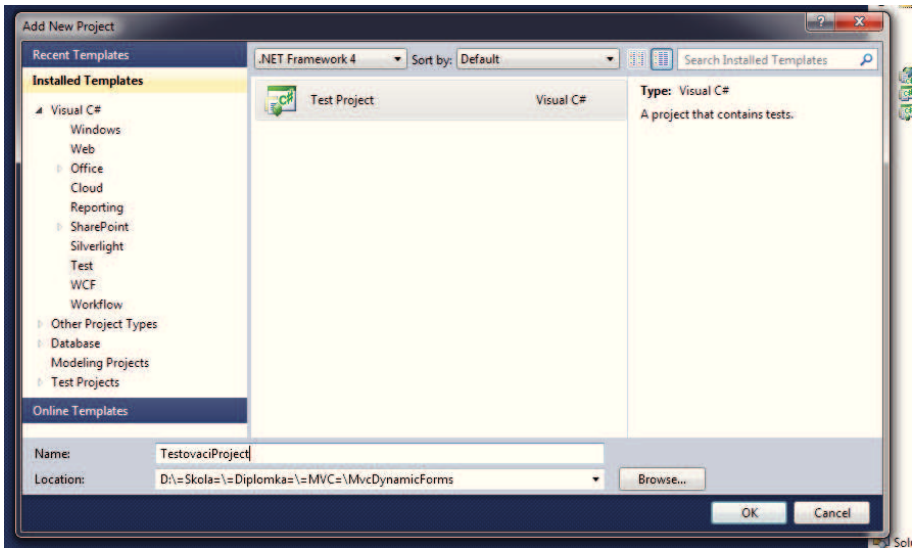
Provedení samotného testování probíhá v internetovém prohlížeči Internet Explorer a vyžaduje mít aktivovaný doplněk „Microsoft Web Test Recorder“. Při testování aplikace je možno využít celé řady testovacích typů.

Mezi nejpoužívanější patří:

- **Smoke test (orientační test)-** Jedná se o krátký a rychlý test, který ověří, jestli je možno aplikaci dále testovat. Typicky dojde k ověření funkčností základních částí systému.
- **Stress test (test zátěže)-** Tento test je využíván k určení stability systému nebo jeho části. Při testování dochází k záměrnému navýšení předpokládaného zatížení systému. Systém je často testován na maximální možný výkon za využití neustále se zvyšujícího počtu uživatelů.
- **Unit test-** testují části komponent nebo modulů. Většinou musí být prováděny přímo programátory, jelikož vyžadují znalosti návrhu systému a zdrojového kódu.
- **Compatibility test-** Tento test slouží k ověření funkčnosti webové aplikace v různých prohlížečích a operačních systémech.
- **Performance test (výkonnostní test)-** Výkonnostní testy jsou jednoduché a poměrně rychlé testy. Obecně slouží k otestování určité části aplikace. Často je v jednom testu využito několik výkonnostních testů, díky tomu můžeme simulovat chování uživatelů.
- **LoadTest (zátěžový test)-** Tento test je zaměřen na otestování výkonu hardwaru, na kterém poběží daná webová aplikace. Jedná se o výkonově náročný test, který simuluje velký počet připojených uživatelů používajících danou aplikaci. Zátěžové testy je nutno pouštět přímo na serveru, protože jenom tak lze mít korektní údaje o zátěži. Také je dobré zátěžový test spouštět v době, kdy lze očekávat minimální přístup uživatelů (typicky v noci), protože požadovaná služba nemusí být plně funkční.

8.1 Testování aplikace

Při vytváření testovací aplikace musíme nejprve vytvořit nový projekt nebo jej přidat k již existujícímu projektu. Při přidávání projektu je nutno vybrat z „Instaled Templatates“ možnost „Test“ a poté „TestProject“, jak je zobrazeno na obrázku 8.



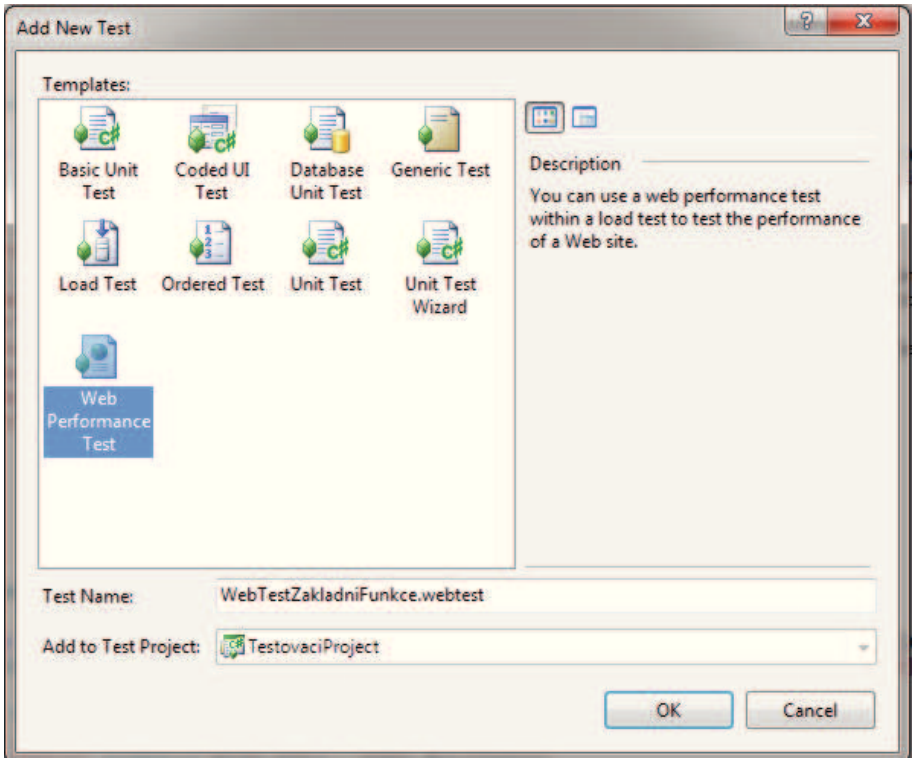
Obrázek 8: Testování - vytvoření projektu

Tímto máme vytvořen testovací projekt a můžeme přistoupit k samotnému vytváření požadovaných testů. Nejprve budeme vytvářet performance testy, které otestují jednotlivé části aplikace.

Pro testování aplikace je třeba vytvořit následující testy:

- Prohlédnutí základních nabídek (přehled vytvoření formulářů, přihlášených uživatelů, atd.)
- Vytvoření akce
- Vytvoření dotazníku
- Přihlášení k formuláři (jak k akci, tak i dotazníku)

Pro vytvoření testu klikneme pravým tlačítkem na název projektu a vybereme „Add ->New Test“. V následující tabulce vybereme „WebPerformanceTest“. Poté jej stačí pojmenovat a test je vytvořen. Vytvoření performance testu je ilustrováno na obrázku 9.



Obrázek 9: Testování - vytvoření testu

Po vytvoření nového testu dojde k otevření internet exploreru, ve kterém je nutno zadat url adresu, kterou chceme testovat. Následně na dané adrese zadáme jednotlivé stránky k otestování a poté zavřeme internet explorer.

Visual Studio poté provede rychlou analýzu zadaných stránek, kdy zjistí, jestli jsou využívány nějaké dynamicky generované parametry.

Posledním krokem je spuštění vytvořeného testu a kontrola, jestli proběhne v pořádku. Pokud nejsou zobrazeny žádné problémy, je možno tento test považovat za úspěšný. Jestliže test skončí chybou, je třeba se podívat na výpis a zjistit, kde nastala chyba, a daný problém vyřešit.

Následně můžeme přejít k vytvoření dalších performance testů, které budou testovat další požadované testy.

8.2 Vytvoření load testu

Jakmile budeme mít vytvořeny všechny performance testy, můžeme vytvořit komplexní test, ve kterém budeme simulovat přístup většího počtu uživatelů.

Pro tento účel je třeba přidat nový Load Test. Při vytváření Load testu dojde ke spuštění průvodce, který umožní snadné vytvoření testu.

Vytvoření Load testu lze popsat v následujících krocích:

1. Pojmenování testu
2. Určení, jestli chceme použít konstantní počet uživatelů nebo budeme postupně navyšovat počet uživatelů
3. Určení, jak mají být jednotlivé testy spouštěny
4. Přiřazení jednotlivých mini testů s možností procentuálního rozdělení využití jednotlivých testů
5. Určení datových připojení (použitá technologie, rychlost, atd.)
6. Definice internetových prohlížečů, kterými bude testování simulováno
7. Možnost definování počítačů, které budou test provádět
8. Určení délky trvání testu a počet iterací
9. Zahájení testování

8.2.1 Nastavení testu

V našem případě budeme testovat aplikaci po dobu dvou hodin za využití 100 uživatelů. Při testování budeme využívat vytvořené performance testy s následujícím procentuálním rozdělením:

- Základní funkce- 25%
- Vytvoření akce – 12%
- Vytvoření dotazníku- 13%
- Přihlášení na akci- 50%

Také budeme využívat tří nejrozšířenějších prohlížečů (Internet Explorer, Firefox, Chrome), kdy každý bude mít 33% zastoupení.

8.2.2 Testovací hardware

Pro testování bylo využito lokálního počítače s následujícími hardwarovými parametry:

- Procesor Intel Core i5-2450M (2x 2.5 GHz)
- 6GB RAM

8.2.3 Parametry serveru

Systém běží na školním serveru, který má tyto hardwarové parametry:

- Procesor Intel Pentium 4, 3.0 GHz
- 3.24 GB Ram

8.2.4 Vyhodnocení testování

Po ukončení testu, dojde k zobrazení přehledu, který obsahuje množství informací. Mezi nejdůležitější pravděpodobně patří možnost zobrazení nejrůznějších grafů a přehledů.

V našem případě jsem vybral několik zajímavých údajů, shrnutých v tabulce 4.

Počet uživatelů	100
Doba trvání (hod:min)	2:10
Počet stránek za sekundu	32.9
Počet požadavků za sekundu	91.7
Průměrná doba načtení stránky (sek.)	1.68
Průměrná doba odpovědi na požadavek (sek.)	0.72

Tabulka 4: Testování - vyhodnocení

Z výše uvedené tabulky vyplývá, že i při konstantním zatížení 100 uživatelů odpovídá systém bez větších obtíží. V opačném případě by bylo nutné provést optimalizaci zdrojového kódu nebo zvýšení výkonu serveru, na kterém je umístěna aplikace.

Pro testování bylo využito 100 uživatelů z důvodu rozumného poměru mezi zatížením testovacího počítače a počtem uživatelů, kteří budou aplikaci využívat. Pokud bychom nastavili test například na 1000 uživatelů, tak by výsledky mohly být ovlivněny výkonem testovacího počítače a neodpovídaly by skutečnému stavu.

V příloze F je umístěn graf průměrné doby načtení stránky a v příloze G je graf zobrazující počet zobrazených stránek za sekundu.

9 Závěr

Cílem této diplomové práce bylo analyzovat a implementovat informační systém pro tvorbu dynamických formulářů za využití technologie ASP.NET MVC3. Tento informační systém by měl být využíván pro účely IT Academy na katedře informatiky a je v současné době dostupný na adrese http://formsmvc.cs.vsb.cz .

Při realizaci této práce bylo využito znalostí nabytých v již absolvovaných předmětech, tak i poznatků získaných z dalších zdrojů. Převážně se jednalo a webové stránky a knihy s danou problematikou. Nakonec byly úspěšně implementovány všechny body zadání a některé časti byly vytvořeny nad jeho rámec.

Do budoucna by bylo možné rozšířit systém o větší integraci se sociálními sítěmi, jako je Facebook nebo Google+. Systém by mohl po vytvoření nového formuláře automaticky vložit odkaz na hlavní stránku uživatele, který ji vytvořil. Také by bylo možné využít technologie, které by vedly ke zlepšení uživatelského vzhledu a implementace nových funkcí (AJAX, podpora exportu do více formátů, atd.).

10 Reference

[1] Jon Galloway, Phil Haack, Scott Hanselman, Scott Guthrie, Rob Conery, *Proffesional ASP. NET MVC2*, WROX ISBN 978-0-470-64318-1

[2] Jon Galloway, Phil Haack, Brad Wilson, K. Scott Allen, *Proffesional ASP. NET MVC3*, WROX ISBN 978-1-119-07658-3

[3] Steven Sanderson, Adam Freeman, *Pro ASP.NET MVC 3 Framework*, Apress ISBN 978-1-4302-3404-3

[4] Dino Esposito, *Programming Microsoft ASP.NET MVC*, Microsoft Press ISBN 978-0-7356-2714-7

[5] Bill Evjen, Scott Hanselmannn, Davin Rader, *Professional ASP.NET 4 in C# and VB*, WROX ISBN 978-0-470-50220-4

[6] ASPNET.CZ,[online] leden 2013. Dostupné z <http://www.aspnet.cz>

[7] ASP.NET, leden 2013. Dostupné z <http://www.asp.net>

[8] Django, duben 2013. Dostupné z <https://www.djangoproject.com/>

[9] Facebook developers, leden 2013.
Dostupné z <https://developers.facebook.com/>

[10] Live Connect Developer Center, leden 2013.
Dostupné z <https://manage.dev.live.com>

[11] MonoRail, duben 2013. Dostupné z <http://www.castleproject.org/projects/monorail/>

[12] Microsoft MSDN, duben 2013. Dostupné z <http://msdn.microsoft.com/en-US>

[13] THING-MODEL-VIEW-EDITOR, duben 2013.
Dostupné z <http://heim.ifi.uio.no/trygver/1979/mvc-1/1979-05-MVC.pdf>

[14] OAuth, duben 2013. Dostupné z <http://oauth.net/>

[15] Python, duben 2013. Dostupné z <http://www.python.org/>

[16] Ruby on Rails, duben 2013.
Dostupné z <http://www.root.cz/serialy/ruby-on-rails/>

[17] StackOverflow, duben 2013. Dostupné z <http://stackoverflow.com>

[18] ASP.NET WEBLOGS, duben 2013. Dostupné z <http://weblogs.asp.net>

[19] w3schools, duben 2013. Dostupné z <http://www.w3schools.com>

[20] Zend, duben 2013. Dostupné z <http://www.zend.com/en/>

A Produkční nasazení

Produkční nasazení je posledním krokem při vývoji aplikace, kdy dojde k přenesení lokálně vyvíjené aplikace na server.

Abychom mohli aplikaci správně využívat, musí server splňovat tyto základní požadavky:

- Podporovat ASP.NET 4.0 a MVC3
- Podporovat SQL databázi

Pro správné fungování aplikace je nutné upravit soubor web.config, umístěný v hlavním adresáři aplikace.

Nejprve je třeba upravit ConnectionString, který definuje umístění databáze. Pro každý server nebo hosting je ConnectionString jiný, proto je třeba správnou cestu zjistit u poskytovatele.

Na výpisu 29 je uveden příklad ConnectionStringu, ve kterém obsahuje několik parametrů. Pro správnou funkčnost je třeba upravit paramert „Data source“ a „AttachDbFilename“.

Data source obsahuje jméno serveru serveru a parametr „AttachDbFilename“ definuje, kde je daná databáze uložena.

```
<add name="ConnectionString" connectionString="
        Data Source=.\SQLEXPRESS;
        AttachDbFilename=|DataDirectory|\Database.mdf;
        Integrated Security=True;
        User Instance=True"
        providerName="System.Data.SqlClient"
/>
```

Výpis 29: Úprava ConnectionString

Tímto je zajištěno propojení s databází a dalším krokem pro správné fungování celé aplikace je úprava emailové adresy a SMTP serveru, přes který jsou odesílány elektronické zprávy. Proto je třeba upravit parametry „emailAdress“ a „smtp“, který je umístěné souboru web.config v sekci „appSettings“. Tyto parametry je třeba upravit dle nastavení serveru, na kterém je aplikace nahrána. Pokud budeme využívat autorizace pomocí Facebooku a Windows Live, je nutno upravit proměnné „FacebookAppID“, „WindowsLiveAppID“ a „appSettings“.

V našem případě jsou parametry uvedeny ve výpisu 30.

```
<appSettings>
    <add key="emailAdress" value="forms@vsb.cz"/>
    <add key="smtp" value="smtp.vsb.cz"/>
    <add key="FacebookAppID" value="1234567890"/>
  <add key="WindowsLiveAppID" value="9876543210"/>
  <add key="WindowsLiveRedirectUrl" value="www adresa"/>
</appSettings>
```

Výpis 30: Úprava nastavení odchozích zpráv

Posledním krokem při instalaci systému je změna hesla u výchozího uživatelského účtu. Účet má přihlašovací jméno a heslo shodné „admin“. Tento uživatel má administrátorská práva a je proto nezbytně nutné změnit heslo na jiné. Poté lze vytvořit libovolný účet, kterému je možno přiřadit administrátorská práva.

B LDAP autorizace

```
public static UserData LDAPAuthorization(string username, string password)
{
    using (LdapConnection con = new LdapConnection(new
        LdapDirectoryIdentifier("ldap.vsb.cz", 636)))
      {
          con.SessionOptions.SecureSocketLayer = true;
          con.AuthType = AuthType.Anonymous;
          con.Bind();
          SearchRequest request = new SearchRequest("",String.Format("(&(
              objectClass=Person)(uid={0}))",username),SearchScope.
              Subtree);

          SearchResponse response = (SearchResponse)con.SendRequest(
              request);
          if (response.Entries.Count == 0){
              return null;
          }
          else{
              SearchResultEntry entry = response.Entries[0];

              string dn = entry.DistinguishedName;

              con.Credential = new NetworkCredential(dn, password);
              con.AuthType = AuthType.Basic;
              try
              {   con.Bind();
                  UserData ud = new UserData();

                  ud.ID = MethodsUser.GetUserIDByLogin(username, "
                      LDAP");
                  ud.authorizationType = "LDAP";
                  ud.login = entry.Attributes["uid"][0].ToString();
                  ud.Name = entry.Attributes["givenname"][0].ToString();
                  ud.Surename = entry.Attributes["sn"][0].ToString();
                  ud.mail = entry.Attributes["mail"][0].ToString();
                  ud.password = "";
                  ud.phone = "";
                  ud.Role = 2;
                  return ud;
              }
              catch
              {
                  return null;
              }
          }
      }
  }
```

Výpis 31: LDAP - kompletní kód

C Windows Live autorizace

```
<script src="https://js.live.net/v5.0/wl.js" type="text/javascript"></script>
<script type="text/javascript">
    var name;
    var surename;
    var mail;

    WL.init({ client_id: "0000000440A8D15",redirect_uri: "http://formsmvc.cs.vsb.
        cz/Account/LogOn",response_type: "token"});

    WL.Event.subscribe("auth.login", onLogin);

    var session = WL.getSession();

    if (session) {displayGreetigns(session);}
    else {WL.ui({ name: "signin", element: "signInButton" }); }

    function onLogin() {
        var localsession = WL.getSession();
        if (localsession) {displayGreetigns(localsession);}}

    function displayGreetigns(passedSession) {
        WL.api({ path: "me", method: "GET" }, function (response) {
            if (!response.error) {
                document.getElementById("LiveName").value = response.
                    first_name;
                document.getElementById("LiveSurename").value =
                    response.last_name;
                document.getElementById("LiveMail").value = response.
                    emails.account;
            }});
        WL.ui({ name: "signin", element: "signInButton" });
        var tokenEl = document.getElementById("token");
        tokenEl.innerHTML = passedSession.access_token;
    }

    function signInUser() {
        WL.login({ scope: "wl.signin,wl.emails" }, onLogin);
        displayGreetigns(localsession);
    }
    function signUserOut() {
        document.getElementById("greeting").innernHTML = "";
        WL.logout();
    }
</script>
```

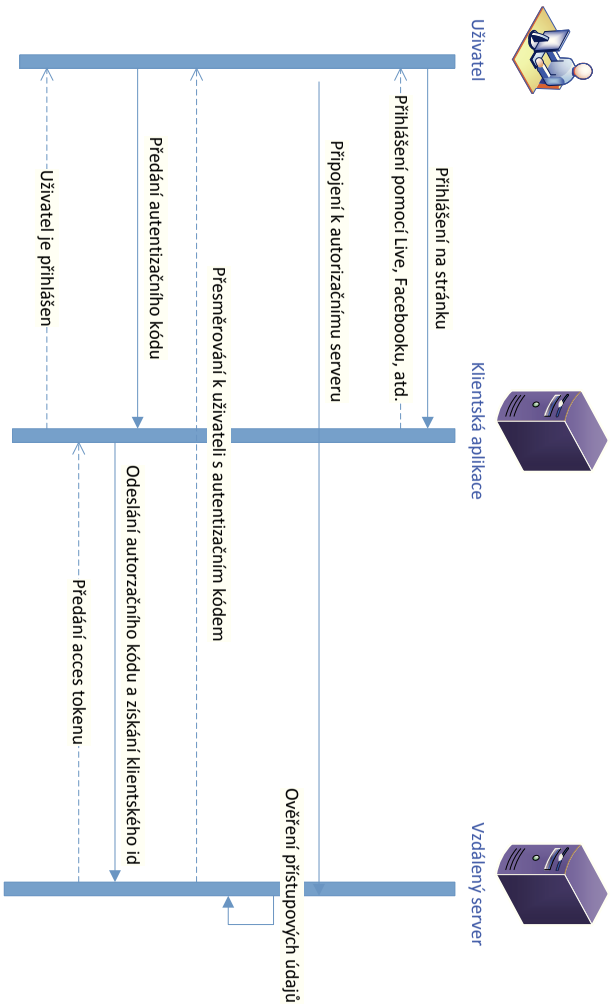
Výpis 32: Windows Live - kompletní kód

D Facebook autorizace

```
<script type="text/javascript"
        src="http://connect.facebook.net/en_US/all.js">
</script>
<script type="text/javascript">
    window.fbAsyncInit = function()
    {FB.init({ appId: '200418406669174',
                status: true,
                cookie: true,
                xfbml: true});};

    FB.login(function(response) { });
    FB.Event.subscribe('auth.login', function(response) {
        FB.api('/me', function(response) { });
        UserInfo(); });

    FB.logout(function(response) {});
    FB.Event.subscribe('auth.logout', function(response) {
        FB.api('/me', function(response) { });
        UserInfo(); });

    function UserInfo() {
        FB.api('/me', function(response) {
            var query = FB.Data.query('select name,email,pic_square from
                user where uid={0}', response.id);
            query.wait(function(rows) {
                document.getElementById("FbName").value = rows[0].name
                    ;;
                document.getElementById("FbMail").value = rows[0].email;

                document.getElementById("btnFB").click();
            }); }); }
</script>
```

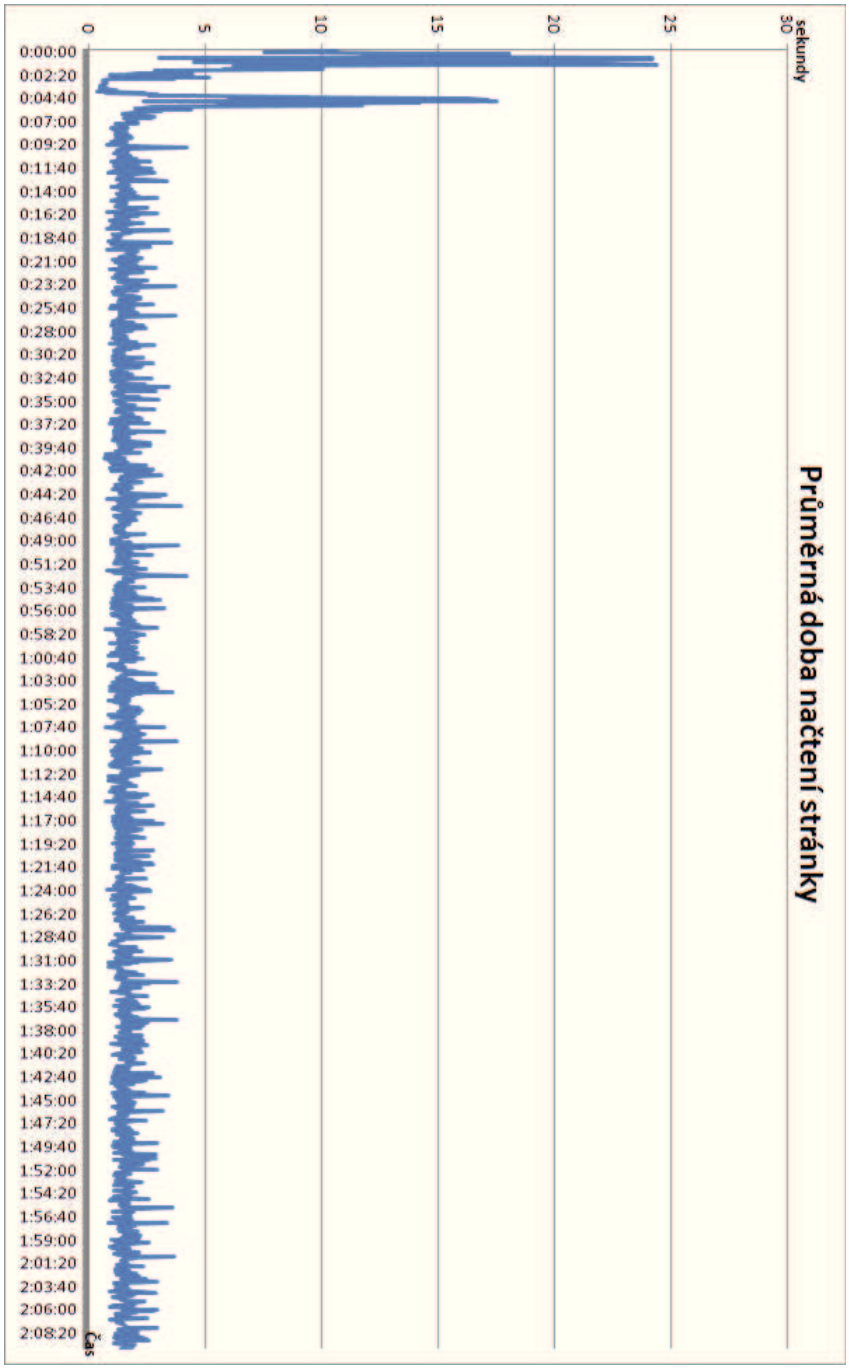
Výpis 33: Facebook - kompletní kód

E OAuth autorizace



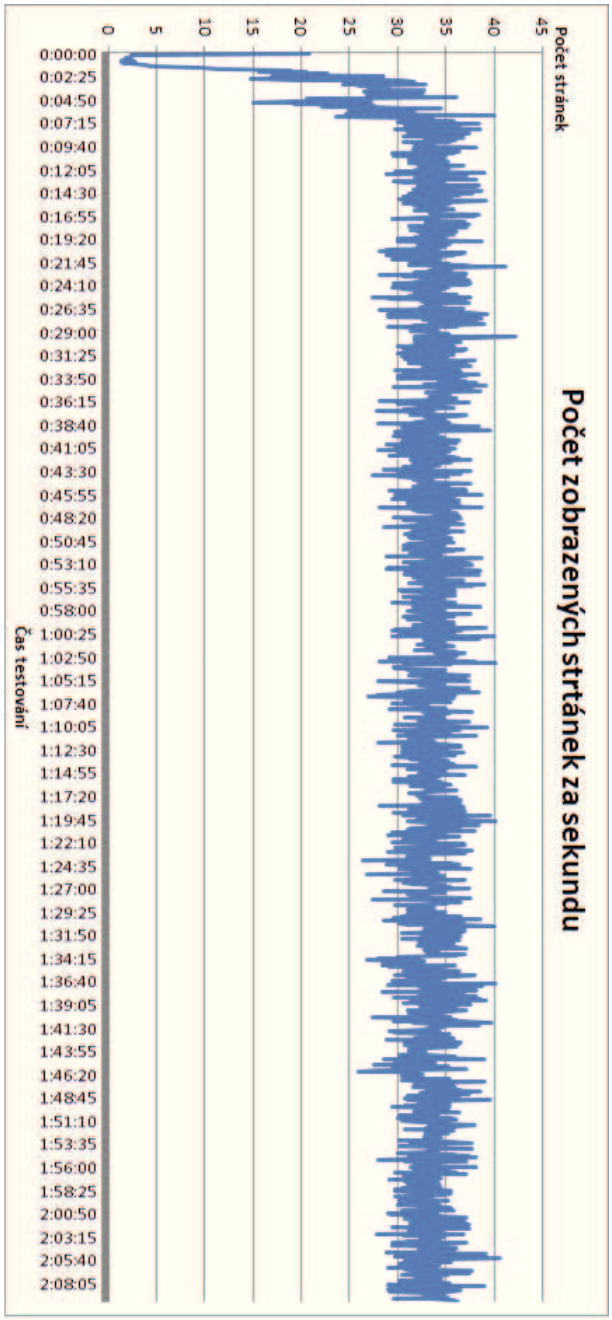
Obrázek 10: OAuth - detailní princip fungování

F Graf průměného načtení stránky



Obrázek 11: Načtení stránek

G Graf počtu zobrazený stránek



Obrázek 12: Počet stránek za sekundu

H Vytvoření ukázkové ASP.NET MVC 3 Aplikace

Pokud chceme vytvářet MVC3 aplikaci, musíme nejdříve splnit následující požadavky:

1. Operační systém Windows XP nebo novější;
2. Visual Studio 2010 nebo 2012 s nadstavbou MVC3.

H.1 Vytvoření nové aplikace

Pro vytvoření nové aplikace musíme provést následující kroky:

1. spustíme Visual Studio a vybereme "File->New->Project";
2. z nabídky vybereme „ASP.NET MVC 3 Web Application", kterou následně pojmenujeme a upřesníme, kde má být uložena;
3. v dalším kroku můžeme upřesnit nastavení a chování aplikace;
4. po stisknutí tlačítka OK dojde k vytvoření nové aplikace, kterou je možno dále upravovat.

Při vytváření aplikace máme na výběr z několika předdefinovaných typů:

1. **prázdný** , tento typ obsahuje pouze základní adresářovou strukturu. Je proto doporučen vývojářům, kteří mají s MVC zkušenosti.
2. **internetová aplikace**, tento typ obsahuje oproti prázdnému template pár základních stránek a jednoduchou autorizaci založenou na ASP.NET Membership. Je vhodný pro všechny uživatele, jelikož po vytvoření je ihned funkční.
3. **intranetová aplikace**, tento typ je velice podobný internetové aplikaci, ale s rozdílem použité uživatelské autorizace.

Další možnosti při vytváření aplikace je upřesnění použitého zobrazovacího enginu a možnosti, jesli má být vygenerován projekt k testování aplikace. Můžeme si vybrat jednak ze starého WebForms enginu, nebo použít nový Razor engine.

H.2 Porozumění aplikační struktury

Po vytvoření nové ASP.NET MVC3 aplikace dojde k automatickému vytvoření několika souborů a adresářů v projektu. Význam jednotlivých adresářů, je popsán v tabulce 5.

Název	Význam
App_Data	Slouží k ukládání souborů, databází, atd.
Content	Slouží k ukládání CSS souborů, obrázků
Controllers	Slouží pro třídy controllerů
Models	Slouží pro třídy reprezentující data a bussines logiku
Scripts	Zde jsou uloženy Javascriptovské soubory
Views	Zde jsou jednotlivé View reprezentující uživatelské rozhraní

Tabulka 5: Význam adresářů

H.3 Jednoduchá aplikace

V jednoduché aplikaci ukážu vytvoření aplikace sloužící k evidenci filmů. Pro jednoduchost budeme evidovat pouze ID, jméno filmu, jméno režiséra a délku. Data budou uložena v SQL databázi a jednotlivé View budou umístěny ve Složce /Views/Home. Při vytváření aplikace musíme vytvořit nový model, který nám bude reprezentovat danou filmovou strukturu. Ten vytvoříme tak, že klikneme pravým tlačítkem na složku "Models->Add New Item" a z nabídky vybereme Class. Pojmenujeme ji například FilmyModel a stiskneme OK.

H.3.1 Model aplikace

Pro definování modelu aplikace vytvoříme nový soubor "FilmyModel.cs" ve složce Models. V tomto souboru následně vytvoříme třídu, která bude reprezentovat strukturu filmu.

U každé možnosti máme definovány jak datové typy, tak i validátory.

```
public class FilmModel
{
    public int Id { get; set; }

    [Required(ErrorMessage = "Je třeba zadat název filmu")]
    [Display(Name = "Jméno filmu:")]
    [MaxLength(150, ErrorMessage = "Jméno může mít maximálně 150
        znaků")]
    public string Jmeno { get; set; }

    [Required(ErrorMessage = "Je třeba zadat jméno režiséra")]
    [Display(Name = "Režisér:")]
    [MaxLength(50, ErrorMessage = "Jméno může mít maximálně 50
        znaků")]
    public string Reziser { get; set; }

    [Required(ErrorMessage = "Je třeba zadat délku filmu")]
    [Display(Name = "Délka filmu")]
    public int Delka { get; set; }

    [Required(ErrorMessage = "Je třeba zadat popis filmu")]
    [Display(Name = "Popis filmu:")]
    [DataType(DataType.MultilineText)]
    public string Popis { get; set; }
}
```

Výpis 34: Ukázková aplikace - model aplikace

H.3.2 Vytvoření controlleru

Pro vytvoření a editaci filmu budeme potřebovat dva controllery. Jeden bude reagovat na metodu GET (vyžádání stránky) a druhý na metodu POST (odeslání stránky).

V prvním controlleru voláme nejdříve metodu ModelState.Clear(), které má za následek to, že se neprovede validace stránky. Pokud bychom tuto metodu nepoužili, došlo by k zobrazení chybových hlášek hned při zobrazení stránky.

Poté zkontrolujeme, jestli je inicializována proměnná TempData["id"], která určuje, jestli budeme vytvářet nový film nebo upravovat již existující. Pokud budeme upravovat již vytvořený film, dojde k načtení informací o filmu, které jsou následně poslány do View.

Druhý controller má za úkol uložení nového filmu do databáze nebo úpravu vytvořeného filmu. V parametru controller je uveden další parametr "string btnSubmit", který je odkazem na tlačítko umístěné ve View a podle toho, které bylo stisknuto, dojde k provedení potřebné operace.

Controllery pro další stránky (seznam filmů, detail filmu) vypadají velice podobně.

```
public  ActionResult NovyFilm(FilmModel model)
{
        ModelState.Clear();

        if  (TempData["id"] !=  null )
        {
                model = ClassDB.DetailyFilmu(Convert.ToInt32(TempData["id"])
                    );
                TempData.Keep("id");
                return  View(model);
        }
        return  View();
}

[HttpPost]
public  ActionResult NovyFilm(FilmModel model,string btnSubmit)
{
     if  (btnSubmit=="Přidat" && ModelState.IsValid)
     {
          ClassDB.PridaniFilmu(model);
          return  RedirectToAction("SeznamFilmu");
     }

     if  (btnSubmit == "Upravit" && ModelState.IsValid)
     {
          model.Id = Convert.ToInt32(TempData["id"]);
          ClassDB.UpravaFilmu(model);
          TempData.Clear();
          return  RedirectToAction("SeznamFilmu");
     }

        return  View(model);
}
```

Výpis 35: Ukázková aplikace - model aplikace

H.3.3 Vytvoření View

View obsahuje základní prvky k zadání informací o filmu a dvě tlačítka, kdy jedno slouží pro vytvoření a druhé k úpravě filmu.

Podobným způsobem jsou vytvořeny stránky DetailFilmu a SeznamFilmu.

```
@model SeznamFilmu.Models.FilmModel

@{
    ViewBag.Title = "Přidání filmu";
 }

@using (Html.BeginForm()){
        <fieldset>
                <legend>Přidání nového filmu</legend>
                <div class="editor-label">
                        @Html.LabelFor(x=>x.Jmeno)
                </div>
                <div class="editor-field">
                        @Html.EditorFor(x=>x.Jmeno)
                        @Html.ValidationMessageFor(x=>x.Jmeno)
                </div>
                <div class="editor-label">
                        @Html.LabelFor(x=>x.Reziser)
                </div>
                <div class="editor-field">
                        @Html.EditorFor(x => x.Reziser)
                        @Html.ValidationMessageFor(x => x.Reziser)
                </div>
                <div class="editor-label">
                        @Html.LabelFor(x=>x.Delka)
                </div>
                <div class="editor-field">
                        @Html.EditorFor(x => x.Delka)
                        @Html.ValidationMessageFor(x => x.Delka)
                </div>
                <div class="editor-label">
                        @Html.LabelFor(x=>x.Popis)
                </div>
                <div class="editor-field">
                        @Html.EditorFor(x => x.Popis)
                        @Html.ValidationMessageFor(x => x.Popis)
                </div>
                <p>
                        @if (TempData["id"] != null){
                                <input type="submit" name="btnSubmit" value="
                                    Upravit"/>
                        }
                        else{
                                <input type="submit" name="btnSubmit" value="Přidat
                                    "/>
                        }
                </p>
        </fieldset>
}
```

Výpis 36: Ukázková aplikace - vytvoření View

I Obsah CD

Na přiloženém CD jsou uložena následující data:

- text práce;
- zdrojový kód systému;
- ukázková aplikace.